Fachhochschule Köln
Cologne University of Applied Sciences

Entwicklung einer Anwendung zum Zeichnen von Entity Relationship Diagrammen im Rahmen der Lernplattform edb

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

vorgelegt von
Noah Ispas

im Studiengang allgemeine Informatik

an der
Fachhochschule Köln, Campus Gummersbach
Fakultät für Informatik und Ingenieurwissenschaften

Erster Prüfer: Prof. Dr. Heide Faeskorn-Woyke
Zweiter Prüfer: M. Sc. Andre Kasper

Gummersbach, 17. Februar 2014

Inhaltsverzeichnis

Kurzzusammenfassung

Im Rahmen dieser Arbeit soll eine Java-Anwendung zum Zeichnen von Entity-Relationship-Diagrammen entwickelt und vorgestellt werden. Die Anwendung soll zur Unterstützung der Veranstaltung Datenbanken an der Fachhochschule Köln - Campus Gummersbach dienen. Der Hintergrund dazu ist die Tatsache, dass im Rahmen der Veranstaltung oftmals Entity-Relationship-Diagramme von Studenten gezeichnet werden sollen. Die Erfahrung der Lehrbeauftragten hat gezeigt, dass Studenten teilweise mit den empfohlenen Anwendungen Schwierigkeiten haben, unter anderem weil diese viel mehr Funktionen anbieten, als von den Studenten benötigt. Eine weitere Schwierigkeit ist die Tatsache, dass verschiedene Anwendungen zum Zeichnen von Entity-Relationship-Diagrammen verschiedenartige Diagramme erzeugen.

Daher besteht der Bedarf einer Anwendungen, die sich einfach bedienen lässt, sich auf das Wesentliche beschränkt und einheitliche Diagramme erzeugt. Die zu entwickelnde Anwendung soll zukünftig innerhalb der Lernplattform edb von Studenten heruntergeladen werden können.

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Im Rahmen der Veranstaltung Datenbanken an der Fachhochschule Köln am Campus Gummersbach müssen Studenten über das Semester verteilt, mehrere fachlich relevante Aufgaben lösen. Die Aufgaben behandeln jeweils verschiedene Themenbereiche von Datenbanksystemen. Die Lösungen müssen von Studenten an bestimmten Pflichtterminen präsentiert und erläutert werden können. Diese sogenannten Praktika dienen dazu, das theoretisch erlernte Wissen der Vorlesung, praktisch umsetzen zu können. Darüber hinaus werden Studenten nur zu der entsprechenden Klausur zugelassen, falls dieses Praktikum erfolgreich absolviert wurde.

Um Studenten eine Hilfestellung zu der Veranstaltung zu geben, existiert zum einen das Buch Datenbanksysteme (vgl. [FWBRB66]), das von den Professorinnen der Veranstaltung (Prof. Dr. Heide Faeskorn-Woyke und Prof. Dr. Birgit Bertelsmeier) geschrieben wurde und die Vorlesung ergänzt. Zum anderen wurde vor einigen Jahren die Lernplattform edb[1] (kurz für eLearning Datenbank Portal) von Nico Liß, Damian Gawenda und Andre Kasper entwickelt. Die Lernplattform beinhaltet unter anderem ein Wiki[2], indem viele Information bezüglich Datenbanksystemen festgehalten werden und ein Forum, in dem sich Studenten zu dem Thema austauschen können. Darüber hinaus beinhaltet die Lernplattform mehrere Anwendungen, die es Studenten erlauben, verschiedene Themenbereiche, meist interaktiv, zu lernen. Unter diesen Anwendungen befindet sich beispielsweise ein Multiple-Choice-Test[3], in dem verschiedene Fragen rund um das Thema Datenbanksysteme gestellt werden und als Übung dann gelöst werden sollen. Des Weiteren existieren viele andere interaktive Anwendungen, wie z. B. ein 3NF-Trainer um zu üben, wie man ein Datenbankschema[4] in die dritte Normalform[5] überführt, und verschiedene Anwendungen zum Üben der Konstruktion von SQL-Select-Statements[6].

[1] vgl. [GLK]

[2] Ein Bereich, in dem Seiten von Anwendern angelegt und editiert werden können, wie auf Wikipedia (vgl. [wika]).

[3] Ein Test, bei dem mehrere Antworten zur Auswahl stehen.

[4] Das Datenbankschema beschreibt die Struktur der Tabellen innerhalb einer Datenbank.

[5] Die Normalformen beschreiben bestimmte Formen eines Datenbankschemas.

[6] SQL ist die Notationsform für die Definition, das Bearbeiten und Abfragen von Daten-

Im Rahmen der bereits erwähnten Praktika müssen Studenten oftmals Entity-Relationship-Diagramme[7] zeichnen und Datenbanken mit mehreren Tabellen und Beziehungen erzeugen. Entity-Relationship-Diagramme vereinfachen die Entwicklung von Datenbanken dahingehend, dass sie ein Modell der zu entwickelnden Datenbank repräsentieren. Ähnlich wie ein Gebäude-Architekt zuerst einen Architekturplan erzeugt, bevor ein Gebäude dann tatsächlich gebaut wird und ein Software-Architekt zunächst UML-Diagramme[8] erzeugt bevor die Software tatsächlich realisiert wird, so zeichnet der Datenbank-Architekt zunächst ein Entity-Relationship-Diagramm. Ein wichtiger Vorteil davon ist die Tatsache, dass die Kommunikation anhand eines Diagramms wesentlicher einfacher ist, als die Kommunikation anhand von SQL-Code. Ein weiterer Vorteil der Modellierung ist, dass alle gängigen Anwendungen, seien es Anwendungen zum Modellieren von UML-Diagrammen oder Anwendungen zum Modellieren von Entity-Relationship-Diagrammen, es erlauben, entsprechenden SQL-Code aus den Diagrammen zu erzeugen.

Die Erfahrung der Lehrbeauftragten der Veranstaltung Datenbanken hat gezeigt, dass oftmals Schwierigkeiten entstehen in Bezug auf das Modellieren von Entity-Relationship-Diagrammen. Im Wesentlichen gibt es dafür zwei Gründe. Ein Grund sind Schwierigkeiten der Studenten bezüglich der Bedienung von Anwendungen zum Modellieren von Entity-Relationship-Diagrammen. Die meisten Anwendungen in dem Bereich (ERwin[9], DBDesigner[10], Microsoft Visio[11] etc.) bieten eine Menge an Funktionalitäten an und sind sehr flexibel bezüglich der Modellierung. Die Erfahrung des Autors hat gezeigt, dass sich sowohl eine hohe Flexibilität als auch eine zu große Menge an Funktionalitäten in Bezug auf Software negativ auf die Verständlichkeit auswirkt. Da Studenten zum Modellieren von Entity Relationship nur wesentliche Funktionen benötigen, sind sie oftmals mit den mächtigen Anwendungen überfordert.

Der zweite Grund betrifft weniger die Studenten, als die Lehrbeauftragten der Veranstaltung Datenbanken. Da mehrere Notationsformen bezüglich

---

bankinhalten. Select-Statements sind Strukturen zum Abfragen bestimmter Datenbankinhalte

[7] Entity-Relationship-Diagramme werden in Abschnitt 2 noch genauer erklärt.

[8] Spezifikation für Diagramme einer Softwarearchitektur, vgl.[Obj].

[9] vgl. [CA].

[10] siehe [fab].

[11] siehe [Mic].

eines Entity-Relationship-Diagramms existieren[12] und verschiedene Anwendungen zusätzlich dazu ihre eigene Form der Darstellung besitzen, entstehen durch die Verwendung verschiedener Anwendungen durch die Studenten auch viele verschiedene **nicht** einheitliche Entity-Relationship-Diagramme. Diese Tatsache erschwert es den Lehrbeauftragten, die Aufgaben im Rahmen des Praktikums bezüglich der Modellierung von Entity-Relationship-Diagrammen, abzunehmen.

Es besteht somit der Bedarf einer Anwendung, die sich durch Begrenzung auf die wesentlichen Funktionalitäten einfach bedienen lässt und einheitliche Diagramme erzeugt. Eine solche Anwendung wird im Rahmen dieser Arbeit entwickelt wie auch dokumentiert und soll nach ihrer Fertigstellung innerhalb der Lernplattform edb als Download angeboten werden.

1.2 Ziele und Zielgruppe

Ziel Das Ziel dieser Arbeit ist es, eine Standalone-Java-Anwendung[13] in Form eines Prototypen zu entwickeln, die es ermöglicht Entity-Relationship-Diagramme zu zeichnen, als Unterstützung der Veranstaltung Datenbanken und als Erweiterung der Lernplattform edb. Dazu gehört die entsprechende Dokumentation der Anforderungen, des Architekturentwurfs, der verwendeten Technologien und Tools sowie die Möglichkeit, das System zu erweitern.

Ein nennenswertes Teilziel ist es hierbei, verschiedene Java-Frameworks zur Visualisierung von Daten zu evaluieren und festzustellen, ob ein solches Framework im Rahmen der Entwicklung verwendet werden kann.

Zielgruppe Die Zielgruppe der Anwendung sind in erster Linie Studenten der Informatik der Fachhochschule Köln am Campus Gummersbach, welche die Veranstaltung Datenbanken besuchen. Die zweite Gruppe sind die Lehrbeauftragten der Veranstaltung Datenbanken, die ggf. die Anwendung verwenden werden um Entity-Relationship-Diagramme zu zeichnen, beispielsweise für Aufgabenstellung eines Praktikums.

[12] Einige der Notationen werden in Abschnitt 2.2 erläutert. Für eine Übersicht aller gängigen Notationsformen, sei auf [Wikb] verwiesen.

[13] Eine Standalone-Anwendung ist eine nicht verteilte Anwendung, die auf einem Computer lokal gestartet wird.

1.3 Gliederung der Arbeit

Diese Arbeit ist wie folgt gegliedert. Zunächst werden Entity-Relationship-Modelle und -Diagramme in Abschnitt 2 eingeführt. Nach der Einführung von Entity-Relationship-Modellen und -Diagrammen werden die Anforderungen an die zu entwickelnde Anwendung in Abschnitt 3 festgehalten. Im Anschluss an die Anforderungen folgt die Evaluation verschiedener Java-Frameworks (Abschnitt 4), die im Rahmen der Entwicklung in Frage kommen, in Bezug auf die funktionalen Anforderungen.

Anschließend wird die Architektur der Anwendung beschrieben. Dazu werden zunächst Entwürfe in Form von Anwendungsfall-, Wire Frame- und Programmablauf-Diagrammen beschrieben (Abschnitt 5). Daraufhin wird die vollständige Architektur der Anwendung über Komponenten- und Klassendiagramme beschrieben (Abschnitt 6) sowie die in der Architektur verwendeten Entwurfsmuster (Abschnitt 7) und verschiedene Möglichkeiten, die Architektur zu erweitern oder anzupassen (Abschnitt 8). Nach der vollständigen Beschreibung der Architektur und der Erweiterungsmöglichkeiten werden die im Rahmen der Entwicklung verwendeten Technologien beschrieben sowie die Möglichkeiten, verschiedene Konfigurationen in dem Zusammenhang vorzunehmen (Abschnitt 9). Abschließend wird die Projektstruktur wie auch einige Screenshots der Anwendung in Abschnitt 10 gezeigt, woraufhin die Schlussbetrachtung folgt (Abschnitt 11).

2 Entity-Relationship-Modell

Da die zu entwickelnde Anwendung das Zeichnen von Entity-Relationship-Diagrammen ermöglichen soll, wird in diesem Abschnitt der Vollständigkeit halber erklärt, was man unter einem Entity-Relationship-Model bzw. -Diagramm versteht (Bezug zu [FWBRB66]).

2.1 Allgemein

Ein Entity-Relationship-Model stellt wie jedes Modell einen Ausschnitt der Realität dar. Es gehört heutzutage zum Standardvorgehen für den Entwurf des Datenbankschemas eines zu entwickelnden relationalen Datenbanksystems. Grund dafür ist, dass es die Struktur einzelner Tabellen[14] (Entitäten)

[14] Attribute samt Datentyp und Schlüsseleigenschaft.

und die jeweiligen Beziehungen[15] zueinander spezifiziert.

Ein Entity-Relationship-Model besteht aus einem Entity-Relationship-Diagramm (visuelle Darstellung des Entity-Relationship-Models, kurz ERD) und einer Beschreibung (in Textform) der einzelnen Elemente.

2.2 Notation

Mit der Zeit haben sich verschiedene Notationen für die visuelle Darstellung eines Entity-Relationship-Diagramms gebildet. Im Folgenden wird die klassische Chen-Notation und die weitverbreitete Krähenfuß-/IE-Notation erläutert:

2.2.1 Chen-Notation

Peter Chen, der übrigens auch das Entity-Relationship-Model eingeführt hat, schlug 1976 die Chen-Notation für die visuelle Darstellung von Entity-Relationship-Diagrammen vor. Abbildung 1 zeigt dazu die Elemente der visuellen Darstellung eines Entity-Relationship-Diagramms entsprechend der Chen-Notation (vgl. [Che76]).

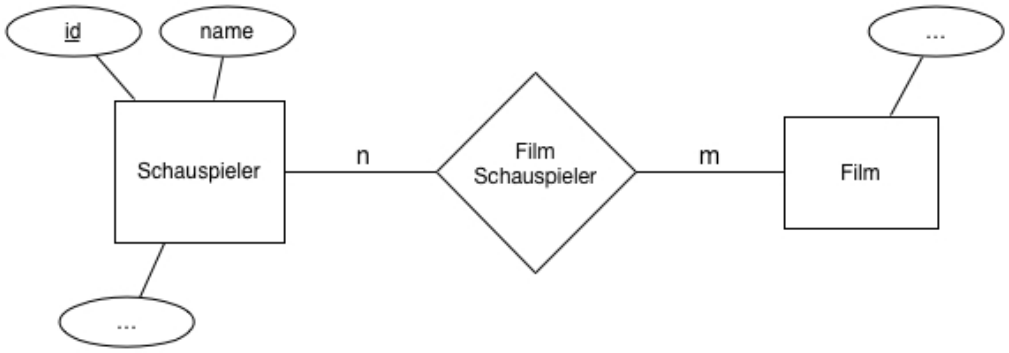


Abbildung 1: *Visuelle Elemente der Darstellung eines Entity-Relationship-Diagramms nach Chen-Notation.*

Die Chen-Notation stellt eine Entität als Rechteck und eine Beziehung als Raute dar. Die Attribute einer Entität sind als ovale Kreise mit der Entität verbunden, wobei Primärschlüsselattribute unterstrichen werden. Die Art der

[15] 1-1-, 1-n-, n-m- und rekursive Beziehungen.

Beziehung, wird durch die jeweilige Kardinalität dargestellt; somit handelt es sich bei der Beziehung aus Abbildung 1 um eine sogenannte n-m-Beziehung.

2.2.2 Krähenfuß-/IE-Notation

Die sogenannte Krähenfuß-, auch IE-Notation (Information Engineering), wurde von James Martin eingeführt (vgl. [Mar59]). Sie beschreibt die Darstellung von Entitäten und Beziehungen zwischen Entitäten, wobei die visuelle Darstellung von Beziehungen je nach Ausprägung an einen Krähenfuß erinnern (siehe Abbildung 2). Eine Beziehung erzeugt immer einen oder mehrere Fremdschlüsseleinträge in der Ziel-Entität mit einer Referenz auf die jeweiligen Primärschlüsselattribute der Quell-Entität der Beziehung. Handelt es sich bei der Beziehung um eine identifizierende Beziehung, so stellt der Fremdschlüssel auch gleichzeitig ein Primärschlüssel dar. Identifizierende Beziehungen werden mit einer durchgezogenen Linie, nicht identifizierende mit einer gestrichelten Linie dargestellt.

Entitäten werden mit in der Krähenfuß-/IE-Notation, als Rechtecke dargestellt, wobei die Attribute innerhalb des jeweiligen Rechtecks visualisiert werden, was an ein Klassendiagramm (UML - Notation) erinnert.

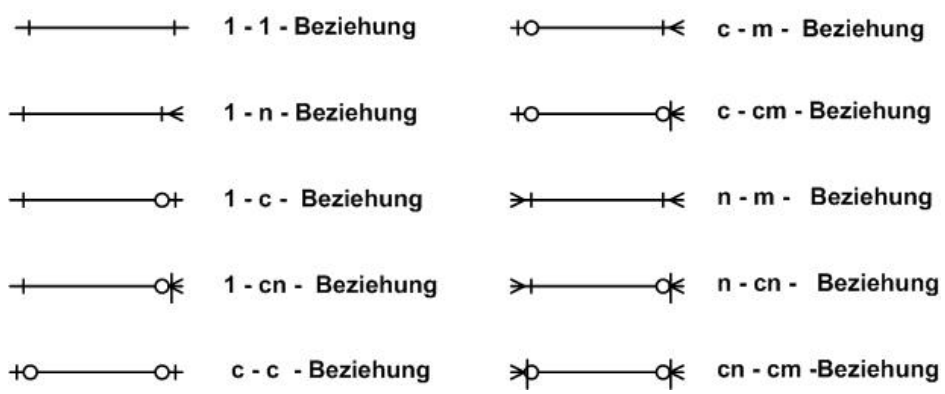


Abbildung 2: *Visuelle Darstellung von Beziehungen zwischen zwei Entitäten entsprechend der Krähenfuß-/IE-Notation (vgl. [FWBRB66]).*

Das Rechteck selbst beinhaltet eine Art Kopfzeile, oder Kopfbereich. Welche Eigenschaften der Entität in der Kopfzeile dargestellt werden, unterscheidet sich je nachdem, welche Anwendung zum Zeichnen des Entity-Relationship-Diagramms verwendet wird. ERwin beispielsweise stellt in der

Kopfzeile die Primärschlüsselattribute dar, wobei die Bezeichnung der Entität über dem Rechteck steht. Dahingegen wird beispielsweise bei der Verwendung der MySQL Workbench[16] die Bezeichnung der Entität in der Kopfzeile dargestellt (siehe Abbildung 3).

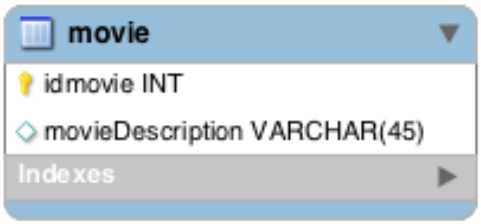


Abbildung 3: *Visuelle Darstellung einer Entität mit der Anwendung „MySQL Workbench“.*

2.3 Erweitertes Entity-Relationship-Model

Das erweiterte Entity-Relationship-Model erweitert die Modellierung relationaler Datenbanksysteme um den Ansatz der Objektorientierung[17] zu einer objektrelationalen Modellierung. Der Motivation dafür ist, dass der relationale Ansatz zwar für die meisten Anwendungen ausreicht, jedoch komplexe Anwendungen oftmals nicht mit relationaler Modellierung abgebildet werden können. Im Folgenden werden die für den Rahmen dieser Arbeit relevanten Aspekte des erweiterten Entity-Relationship-Models erläutert.

2.3.1 Beziehungsarten

Um Beziehungen des relationalen Modells um den Ansatz der Objektorientierung zu erweitern, sieht das erweiterte Entity-Relationship-Model im Wesentlichen zwei neue Arten von Beziehungen vor:

[16] vgl. [Sun]

[17] Die Objektorientierung ist das derzeit dominierende Paradigma in Bezug auf Softwareentwicklung. Sie löst die klassische prozedurale Programmierung ab und erlaubt die Abbildung von Dingen aus der Realität und deren Eigenschaften auf Klassen und deren Attribute.

Vererbung Die Vererbung ist ein Konzept, das die Beziehung von Supertyp und Subtyp beschreibt. Ein Subtyp erweitert einen Supertyp um bestimmte Eigenschaften und kann somit in eine Vererbungsbeziehung mit dem Supertyp gebracht werden. Dabei erbt der Subtyp alle Eigenschaften eines Supertypen. Diese Beziehung wird im Rahmen des erweiterten Entity-Relationship-Modells als “IS-A“-Beziehung bezeichnet.

Ein Konstrukt aus Supertypen und Subtypen, ergibt sich durch Spezialisierung oder Generalisierung. Bei der Spezialisierung, dem deduktiven Ansatz, wird ein gegebener Supertyp um bestimmte Eigenschaften erweitert und repräsentiert somit einen Subtypen. Bei der Generalisierung, dem induktiven Ansatz, werden verschiedene bestehende (Sub-)Typen aufgrund von gemeinsamen Eigenschaften, zu einem Supertyp geformt. Die gemeinsamen Eigenschaften werden dabei in den Supertypen extrahiert (siehe Abbildung 4).

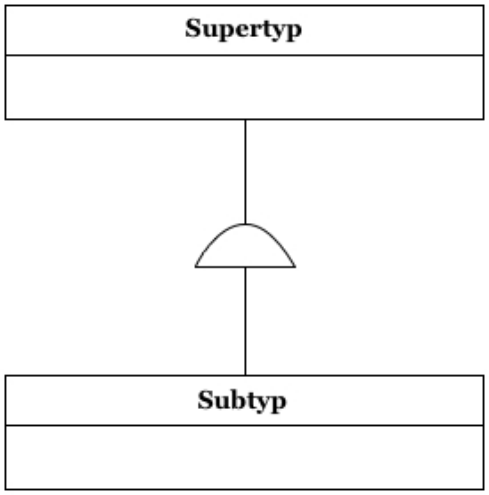
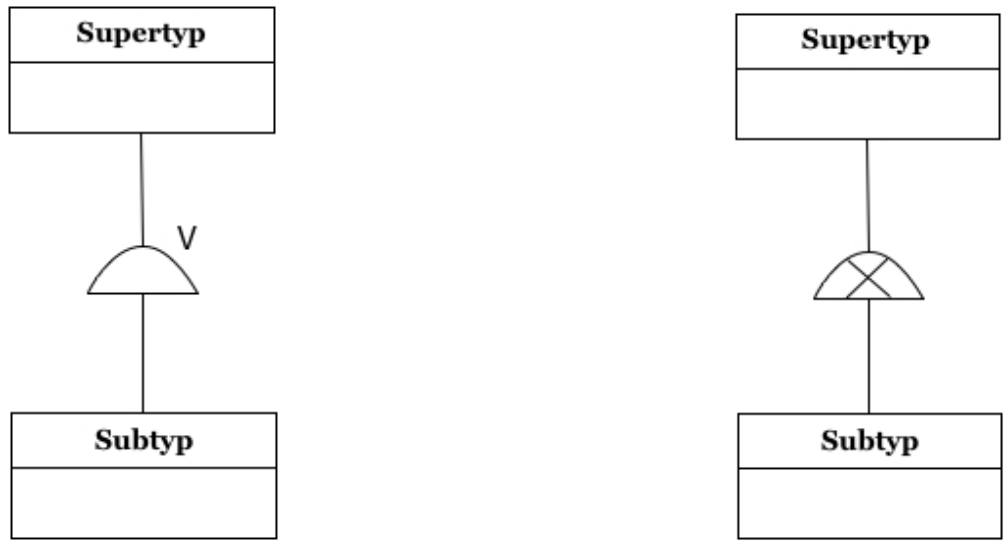


Abbildung 4: *Darstellung einer “IS-A“-Beziehung zwischen einem Subtyp und einem Supertyp.*



(a) (b)

Abbildung 5: *Darstellung von vollständigen und disjunkten "IS-A"-Beziehungen.*

Konstrukte von Supertypen und Subtypen können zusätzlich die Eigenschaften „vollständig" und/oder „disjunkt" besitzen (siehe Abbildung 5). Die Eigenschaft „vollständig" (5a) bedeutet in diesem Zusammenhang, dass der „Supertyp keine eigenen Elemente enthält, also jedes Element in einem Subtypen enthalten ist"(vgl. [FWBRB66]). Die Eigenschaft „disjunkt" (5b) signalisiert, dass „die einzelnen Subtypen keine gemeinsamen Elemente haben" (vgl. [FWBRB66]).

Aggregation Bei der Aggregation handelt es sich um ein Konstrukt, das eine Beziehung zwischen einem Ganzen und einem Teil des Ganzen beschreibt, weshalb sie auch “Ist Teil von“-Beziehung bezeichnet wird. Syntaktisch wird diese Beziehung als 1-n-Beziehung dargestellt.

2.3.2 Eigener SQL-Typ

Im Rahmen der Objektorientierung können nicht nur Dinge aus der Realität auf Klassen abgebildet werden, es können auch logische Konstrukte Erschaffen werden, die in der Realität nicht existieren. Das erweiterte Entity-Relationship-Modell sieht für den Aspekt der Objektorientierung, das Erschaffen neuer SQL-Datentypen vor.

3 Anforderungsmanagement

Am Anfang eines jeden Softwareprojekts steht das Anforderungsmanagement[18]. Hierbei werden die funktionalen und nichtfunktionalen Anforderungen an die Anwendung ermittelt. Im Rahmen dieser Arbeit wurden die Anforderungen zwischen dem Autor der Arbeit und den Prüfern festgelegt und werden im Folgenden aufgeführt.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen beantworten die Frage, welche Funktionen die Anwendung anbieten soll. Die funktionalen Anforderungen sind hierbei in primäre und sekundäre Anforderungen unterteilt, wobei primäre Anforderungen eine primäre Priorität besitzen und die sekundären Anforderungen eine sekundäre Priorität.

3.1.1 Primär

F10 Die Anwendung soll es Nutzern ermöglichen, Entitäten zu zeichnen und zu löschen. Dabei soll es möglich sein, Attribute hinzuzufügen und zu löschen. Die Attribute sollen dahingehend bearbeitet werden können, als dass

[18] Sowohl bei klassischen Vorgehensmodellen der Softwareentwicklung, wie beispielsweise dem Wasserfallmodell, als auch bei moderneren, agilen Vorgehensmodellen wie Scrum (vgl. [Scr]) wird ein Anforderungsmanagement durchgeführt.

die Bezeichnung eines Attributs sowie der jeweilige SQL-Datentyp angegeben werden können. Darüber hinaus kann ein Attribut oder mehrere als Primärschlüssel markiert werden. Abbildung 6 zeigt, wie die Entitäten visuell dargestellt werden sollen.

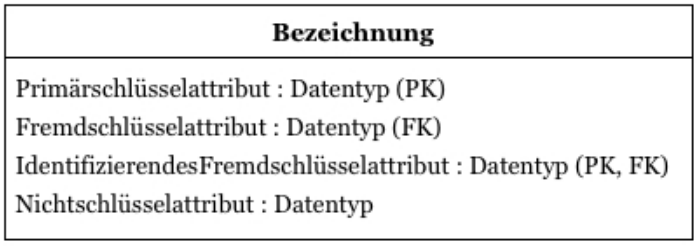


Abbildung 6: *Visuelle Darstellung einer Entität im Rahmen der zu entwickelnden Anwendung.*

F20 Neben dem Zeichnen, Bearbeiten und Löschen einer Entität soll es Anwendern möglich sein, Beziehungen zwischen Entitäten zu zeichnen. Es werden hierbei lediglich binäre Beziehungen zugelassen, also Beziehungen zwischen zwei Entitäten.

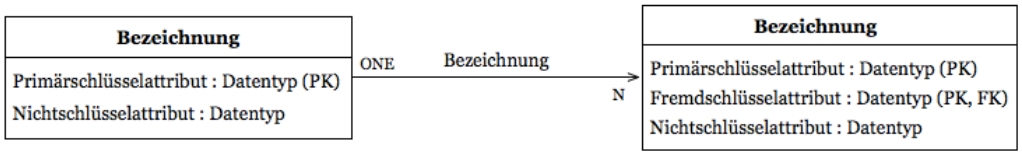


Abbildung 7: *Visuelle Darstellung einer identifizierenden Beziehung zwischen zwei Entitäten.*

Die Anwendung soll dabei erlauben, die Kardinalität der Beziehung zu wählen und auch, ob es sich bei der Beziehung um eine identifizierende (Abbildung 7) oder nichtidentifizierende Beziehung (Abbildung 8) handelt. Dabei wird eine Notation gewählt, bei der die Kardinalitäten an den jeweiligen Enden der Beziehung in Textform dargestellt werden. Der Pfeil wurde eingeführt, um zu zeigen, welches die Quell-Entität und welches die Ziel-Entität der Beziehung ist; die Strichelung der Beziehung signalisiert, dass es sich um eine nichtidentifizierende Beziehung handelt.

Das Zeichnen einer Beziehung erzeugt einen Fremdschlüssel in der Ziel-Entität der Beziehung, was übrigens die einzige Möglichkeit ist, einen Fremdschlüssel in einer Entität anzulegen.

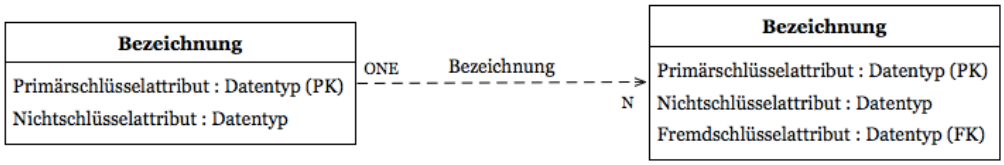


Abbildung 8: *Visuelle Darstellung einer nichtidentifizierenden Beziehung zwischen zwei Entitäten.*

F30 Das statische Zeichnen von Entitäten und Beziehungen reicht nicht aus, um eine Anwendung zu realisieren, mit der man Entity-Relationship-Diagramme erzeugen kann. Anwender sollen ebenfalls die Möglichkeit haben, Entitäten nach eigenem Belieben zu bewegen. Dabei sollen die bestehenden Beziehungen zwischen Entitäten erhalten bleiben und ggf. neu gezeichnet werden.

F40 Um gezeichnete Entity-Relationship-Diagramme persistieren zu können, soll es möglich sein, ein Diagramm in eine Datei zu exportieren. Zudem sollen exportierte Dateien natürlich auch wieder importiert werden können. Für das Format dieser Dateien eignet sich die XML-Technologie[19], da XML-Dateien relativ komfortabel mit Java bearbeitet werden können (siehe JAXB, vgl. [Gla]).

F50 Um ein bereits gezeichnetes Entity-Relationship-Diagramm einsehen zu können, müssten Anwender eine vorher exportierte Datei öffnen. Da dieses Vorgehen für das reine Einsehen eines Diagramms recht umständlich ist, soll es ebenfalls möglich sein, ein Modell in eine JPG-Datei zu exportieren.

F60 Das Zeichnen von Entity-Relationship-Diagrammen hat in erster Linie den Vorteil, dass ein zu entwickelndes Datenbanksystem visuell dargestellt wird. Dies vereinfacht die Kommunikation mit Anderen Menschen sowie Änderungen an der Struktur des zu entwickelnden Datenbanksystems.

[19] vgl. [Wor]

Ein weiterer Vorteil ist die Tatsache, dass aus einem Entity-Relationship-Diagramm, SQL-Code (DDL[20]) generiert werden kann, da alle relevanten Aspekte in dem Modell enthalten sind. Daher soll es die im Rahmen dieser Arbeit entwickelte Anwendung eröglichen, aus dem Modell MySQL- (vgl. [Orab]) und Oracle-SQL-Code (vgl. [Oraf]) zu generieren.

3.1.2 Sekundär

F70 Die Abbildungen 7 und 8 aus der Anforderung F10 zeigen, wie Beziehungen zwischen zwei Entitäten im Rahmen der zu entwickelnden Anwendung visuell dargestellt werden sollen. Die Kardinalität wird dabei als Text am jeweiligen Ende der Beziehung angezeigt. Es wäre wünschenswert, die Beziehungen entsprechend der Krähenfuß-/IE-Notation darzustellen (siehe Abbildung 2).

F80 In Abschnitt 2.3.1 wurden die Beziehungen eines erweiterten Entity-Relationship-Models eingeführt. Da die primären Anforderungen lediglich relationale Beziehungen zwischen Entitäten vorsieht, soll die Anwendung um das Zeichnen von objektrelationalen Beziehungen, gemäß dem erweiterten Entity-Relationship-Model, erweitert werden. Dabei müssen lediglich die Vererbungsbeziehungen ("IS-A" und "Ist Teil von") explizit implementiert werden, da es sich bei Aggregationen – wie bereits erwähnt – syntaktisch um 1-n-Beziehungen handelt; der einzige Unterschied ist semantischer Natur.

F90 Eine weitere funktionale Anforderung ist das Erzeugen und Visualisieren eines eigenen SQL-Datentyps, siehe Abschnitt 2.3.2.

3.2 Nichtfunktionale Anforderungen

Die nicht funktionalen Anforderungen beschreiben grob gesagt die Qualitätsmerkmale eines zu entwickelndes Softwaresystems und werden derzeit in dem ISO-Standart 25010:2011[21] beschrieben. Im Folgenden werden nicht alle, sondern nur die wichtigen an die zu entwickelnde Anwendung gestellten, nichtfunktionalen Anforderungen erläutert.

[20] Die Data Definition Language ist ein Aspekt der SQL-Notationsform, der die Definition von Tabellen innerhalb einer Datenbank betrifft.

[21] vgl. [Int]

N10 Die Anforderung der „Funktionalität“ fordert, dass alle Funktionen gemäß den funktionalen Anforderungen samt geforderten Eigenschaften und auch wie spezifiziert funktionieren. Um diesbezüglich ein Beispiel zu nennen, soll ein von dem Anwender hinzugefügtes Attribut einer Entität, auch innerhalb der grafischen Darstellung angezeigt werden.

N20 Zu dem Aspekt der „Zuverlässigkeit“ in Bezug auf die zu entwickelnden Anwendung als nichtfunktionale Anforderung wird zum einen gefordert, dass eine gewisse Fehlertoleranz bezüglich der Bedienung durch Anwender besteht. Genauer gesagt bedeutet dies, dass kleinere Fehler der Bedienung entsprechend toleriert werden und kritische Fehler auf angemessene Art und Weise abgefangen werden.

Zum anderen wird gefordert, dass die Anwendung auch bei längerer Verwendung wie gefordert funktioniert und kein invalides Verhalten zeigt.

N30 Die nichtfunktionale Anforderung der „Benutzbarkeit“ fordert, dass die Anwendung ohne zusätzlichen Aufwand von Anwendern bedient werden kann. Um dieser Anforderung zu entsprechen, werden für gewöhnlich Methoden aus dem Bereich Usability-Engineering[22] verwendet, um eine Benutzeroberfläche möglichst ergonom zu gestalten.

N40 Der Aspekt der „Änderbarkeit“ betrifft eher die technische Seite der Anwendung. So wird gefordert, dass die bestehende Architektur der Software analysiert, modifiziert und getestet können werden können. Es existieren viele Aspekte, die das Erfüllen dieser Anforderung bekräftigen. Dazu gehört beispielsweise die Dokumentation der Softwarearchitektur sowie eine „saubere[23]“ Implementierung.

[22] vgl. [Ram]

[23] Mit „sauber“ ist hier die Verwendung von bewährten Prinzipien bezüglich des Entwurfs einer Softwarearchitektur gemeint, von denen einige im weiteren Verlauf der Arbeit noch öfter genannt werden.

4 Evaluation verschiedener Java-Frameworks zur Visualisierung von Daten

Der letzte Abschnitt hat die Anforderungen an die zu entwickelnde Anwendung aufgezeigt. In diesem Abschnitt werden nun einige Java-Frameworks[24] evaluiert, die sich für eine Visualisierung von Daten eignen. Dazu werden kleine Anwendungen implementiert, die das jeweilige Framework verwenden. Anhand der Evaluation soll entschieden werden, ob im Rahmen der Entwicklung ein Framework zur Visualisierung benutzt werden kann oder nicht und falls ja, dann welches.

Dazu sei erwähnt, dass die Verwendung eines Frameworks grundsätzlich sowohl einen großen Vor- als auch einen großen Nachteil hat. Der offensichtliche Vorteil ist die Tatsache, dass Frameworks bereits viele Funktionalitäten enthalten, die verwendet werden können, ohne sie erneut implementieren zu müssen. Es kann dadurch viel Arbeit und Zeit gespart werden. In den meisten Fällen bietet ein Framework zwar viele, allerdings nicht alle benötigten Funktionen. Diese Tatsache führt zu dem Nachteil der Verwendung eines Frameworks. Je nachdem, wie komplex und gut dokumentiert das Framework ist, kann es schwer bis unmöglich werden, das Framework an die gegebenen Anforderungen anzupassen oder zu erweitern.

4.1 Graphen-Frameworks

Im Rahmen einer ersten Recherche wurde festgestellt, dass sich für das Zeichnen von Entity-Relationship-Diagrammen sogenannte Graphen-Frameworks eignen könnten. Im Wesentlichen gibt es dafür zwei Gründe:

Ursprünglich kann man mit Graphen-Frameworks, Graphen (vgl. Graphentheorie) visualisieren und teilweise auch interaktiv bearbeiten. Rein visuell betrachtet sind Entity-Relationship-Diagramme einem Graphen, auf abstrakter Ebene, ähnlich. So besteht ein Entity-Relationship-Diagramm aus Entitäten, die ggf. durch Beziehungen miteinander verbunden sind. Ein Graph besteht aus Knoten, die ggf. über Kanten miteinander verbunden sind. Der erste Grund ist somit, dass Entity-Relationship-Diagramme, visuell betrachtet, auf Graphen abgebildet werden können.

Der zweite Grund ist die Tatsache, dass fast alle der Graphen-Frameworks eine Anpassung der visuellen Darstellung von Knoten und Kanten erlauben.

[24] siehe [ITW]

Wie detailliert man dabei die visuellen Darstellung anpassen kann, ist von Framework zu Framework verschieden. Einige Frameworks erlauben die Anpassung der Darstellung ohne Veränderung der eigentlichen Form eines Graphen. Dahingegen kann man mit anderen Frameworks die Form von Knoten und Kanten auf sehr vielfältige, flexible Weise anpassen. Diese Tatsache ist sehr wichtig, da ein Entity-Relationship-Model zwar, abstrakt betrachtet, visuell auf ein Graphen abgebildet werden kann, aber dabei definitiv eine Anpassung der visuellen Darstellung von Nöten ist.

4.2 Bewertungskriterien

Um verschiedene Java-Graphen-Frameworks evaluieren zu können müssen zunächst Bewertungskriterien definiert werden. Anhand der Bewertungskriterien soll entschieden werden, ob ein Graphen-Framework im Rahmen dieser Arbeit verwendet werden soll und falls ja, welches. Die Bewertungskriterien leiten sich dabei unter anderem aus den in Abschnitt 3.1 genannten funktionalen Anforderungen an die zu entwickelnde Anwendung ab und werden im Folgenden aufgeführt:

Open Source Da ein Graphen-Framework in jedem Fall angepasst bzw. erweitert werden muss, sollte es sich bei dem Framework um ein Open Source[25] Framework handeln. Die Möglichkeit, über ein Open-Source-Frameworks einzelne Elemente einsehen zu können, vereinfacht dabei die Anpassbarkeit bzw. Erweiterbarkeit.

Dokumentation Neben der Tatsache, dass einzelne Elemente eines Graphen-Frameworks eingesehen werden können, erleichtert die zusätzliche Dokumentation und die Verwendung von (abstrakten) Java-Klassen und -Schnittstellen die Anpassung, Erweiterung und Verwendung eines Frameworks.

Hilfestellung im Internet Handelt es sich bei einem Framework um ein Open-Source-Framework, dass gut dokumentiert ist, so ist die Anpassung, Erweiterung und Verwendung wesentlich leichter. Allerdings muss man sich intensiv mit den einzelnen Elementen des Frameworks befassen, um es erweitern, anpassen und verwenden zu können. Durch zusätzliche Hilfestellung

[25] vgl. [Ope]

aus dem Internet, in Form von Tutorials oder Code-Beispielen, kann viel Zeit bezüglich der Auseinandersetzung mit dem Framework gespart werden.

Anpassung der grafischen Darstellung Des Öfteren wurde jetzt bereits erwähnt, dass ein Graphen-Framework angepasst bzw. erweitert werden muss, um ein Entity-Relationship-Diagramm zeichnen zu können. Genauer gesagt muss die Form und der Inhalt von Knoten und Kanten verändert werden können. Knoten müssen als Rechtecke dargestellt werden können, entsprechend Abbildung 7. Kanten sollen entsprechend der primären Anforderung F20 zunächst als gestrichelte oder durchgehende Linie, mit einer entsprechenden Bezeichnung, dargestellt werden können. Darüber hinaus sollen die Kardinalitäten der beteiligten Entitäten an dem jeweiligen Ende des Kante als Text dargestellt werden (siehe Abbildung 8). Die grafische Darstellung soll innerhalb eines Java-JPanels[26] möglich sein und es wäre wünschenswert, die Darstellung der Kanten entsprechend der Krähenfuß-/IE-Notation-Notation anpassen zu können (siehe sekundäre Anforderung F70).

Interaktion mit der grafischen Darstellung Neben der visuellen Darstellung eines Entity-Relationship-Diagramms soll es mithilfe eines Graphen-Framework möglich sein, einzelne Knoten anzuklicken und zu bewegen (entsprechend F30). Die Interaktion mit einem Graph soll dabei individuell angepasst werden können, um beispielsweise andere Aktionen mit einem Mausklick zu verknüpfen.

Verknüpfung der grafischen Elemente mit Metadaten Entsprechend dem Entwurfsmuster Model View Controller (wird in Abschnitt 7.1 ausführlich erläutert) und den Design Prinzipien[27] „Separation of Concerns[28]“ sowie „Single Responsibility[29]“ wird der Code für die grafische Darstellung und für den Umgang mit Geschäftsdaten[30] voneinander getrennt. Visuell dargestellte Entitäten oder Beziehungen müssen aber dennoch auf irgendeine Weise mit

[26] Eine Strukturierungshilfe im Rahmen der Programmierung grafischer Oberflächen mit Java.

[27] In Bezug auf Softwarearchitektur.

[28] vgl. [Gre]

[29] siehe [Mar]

[30] Geschäftsdaten sind die elementaren Daten, die im Rahmen einer Anwendung verwendet werden. Für die im Rahmen dieser Arbeit zu entwickelnde Software sind Geschäftsdaten Elemente eines Entity-Relationship-Diagramms.

dem entsprechenden Geschäftsobjekt[31] verknüpft werden. Es wäre vorteilhaft, wenn ein Graphen-Framework die Möglichkeit biete würde, Elemente, die visuell dargestellt werden sollen, mit Metadaten zu verknüpfen. Die Verknüpfung von Elementen mit Metadaten erleichtert die Zuordnung von visuell dargestellten Elementen zu den entsprechenden Geschäftsobjekten. Ein entsprechender Mechanismus muss somit nicht explizit implementiert werden.

4.3 Vergleich

Im Folgenden werden 3 Frameworks miteinander verglichen. Dabei werden die einzelnen Frameworks vorgestellt, Beispiele gezeigt, und anhand der Bewertungskriterien einander gegenübergestellt. Das Ergebnis dieser Gegenüberstellung ist die Entscheidung, ob eines der vorgestellten Frameworks für die Visualisierung des Entity-Relationship-Models verwendet werden kann und falls ja, dann welches.

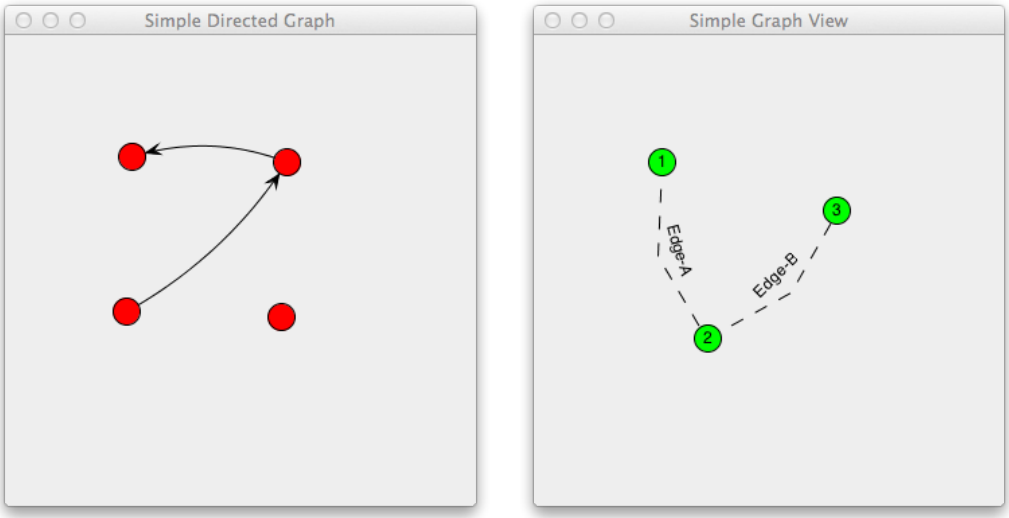
4.3.1 Java Universal Network/Graph Framework

Das Java Universal Network/Graph (kurz JUNG) Framework erlaubt die Visualisierung von Daten, die als Graph (Abbildung 9 (a)) oder Netzwerk dargestellt werden können. Die Architektur des JUNG-Frameworks soll viele Arten der Visualisierung unterstützen, beispielsweise un-/gerichtete Graphen, Multigraphen und Hypergraphen. Darüber hinaus können die einzelnen Elemente mit Metadaten verknüpft werden (vgl. [OFNb]).

Der Quellcode des JUNG-Frameworks kann eingesehen werden, da es sich um ein Open-Source-Framework handelt. Allerdings ist die Dokumentation des Frameworks eher spärlich. Innerhalb des Quellcodes sind einige der wichtigen Elemente nicht ausreichend oder gar nicht dokumentiert. Als Hilfestellung bieten die Entwickler des JUNG-Frameworks daher Beispielcode für die Verwendung einzelner Aspekte des Frameworks sowie eine dazugehörige Dokumentation (vgl. [OFNa]). Durch diese Hilfestellung ist es relativ einfach, einen Graphen zu visualisieren. Falls man das Framework allerdings tiefergehend verändern möchte, lassen sich vergleichsmäßige wenige Beispiele im Internet finden.

Abbildung 9 (b) zeigt, dass die Visualisierung eines Graphen grundsätzlich angepasst werden kann. So können Kanten als gestrichelte oder durchgehende Linie mit Bezeichnung dargestellt werden. Darüber hinaus können

[31] In diesem Fall eine Entität oder eine Beziehung.

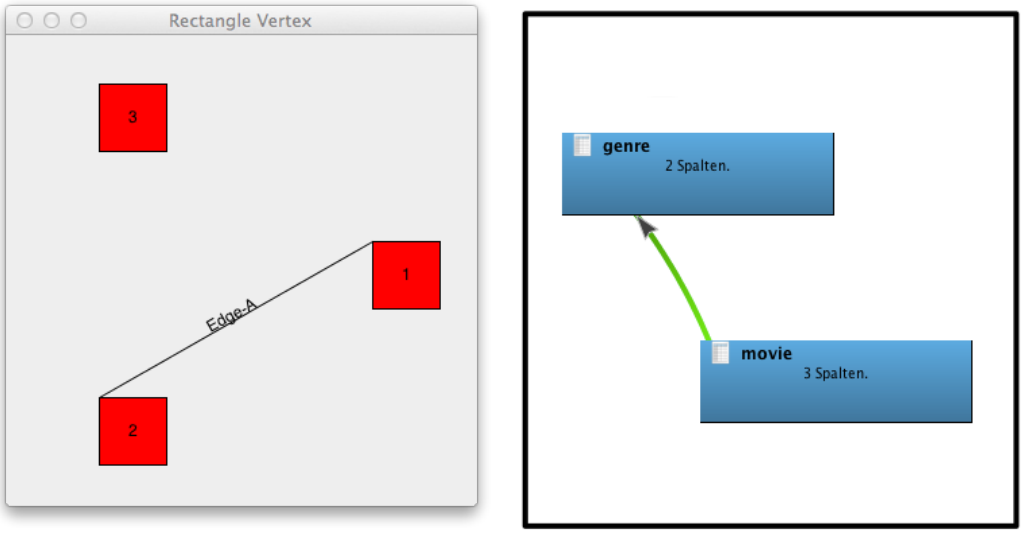


(a) (b)

Abbildung 9: *Einfacher und leicht angepasster Graph.*

Knoten als Rechtecke dargestellt und mit einem mit spezifischem Inhalt versehen werden (Abbildung 10). Ob es möglich ist, eine Kante mit zwei Beschriftungen oder in Form der Krähenfuß-/IE-Notation darzustellen konnte noch nicht explizit verifiziert werden. Bezüglich der Interaktion bietet das JUNG-Framework die Möglichkeit, einzelne Knoten auszuwählen und zu bewegen wobei die Kanten erhalten bleiben.

Zusammenfassend kann gesagt werden, dass sich das JUNG-Framework durchaus für die Darstellung von Entity-Relationship-Diagrammen eignet, da Knoten und Kanten in ihrer visuellen Darstellung angepasst und interaktiv bewegt werden können. Die Dokumentation und Hilfestellungen im Internet lassen zwar zu wünschen übrig, reichen jedoch aus, um das Framework den eigenen Bedürfnissen entsprechend anpassen bzw. erweitern zu können.



(a) (b)

Abbildung 10: *Graph mit angepasster Darstellung, wobei Abbildung (b) ein Ausschnitt der Anwendung „visualdependencies“ zeigt, die im Rahmen einer Diplomarbeit entwickelt wurde und das JUNG-Framework verwendet(siehe [KP09]).*

4.3.2 JGraphX

JGraphX ist ein Framework, welches ebenfalls für die Visualisierung von Graphen verwendet werden kann (Abbildung 11). Es handelt sich dabei um die sechste Version des Frameworks JGraph. Der Name wurde geändert, da die sechste Version von Grund auf neu entwickelt wurde. Mithilfe des JGraphX-Frameworks soll es möglich sein, verschiedene Modellierungstools zu implementieren (vgl. [JGrb]).

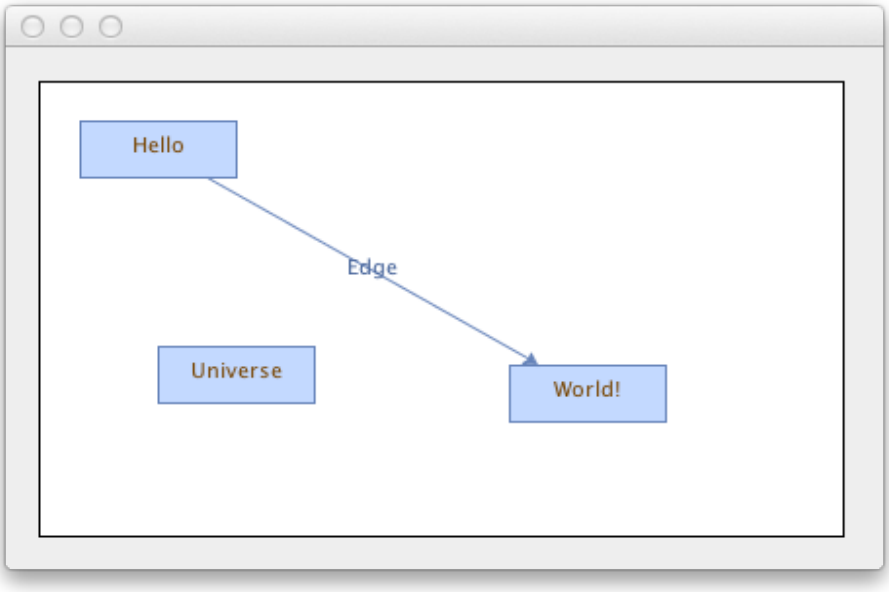


Abbildung 11: *Beispielanwendung zeigt die Standard-Darstellung eines Graphen mit dem JGraphX-Framework.*

Das JGraphX-Framework ist ein Open-Source-Framework somit kann der Quellcode wie bei dem JUNG-Framework eingesehen werden. Nach der Implementierung einer Beispielanwendung für die Evaluation kann gesagt werden, dass die Dokumentation relativ gut ist, jedenfalls besser als die Dokumentation des JUNG-Frameworks. Es existiert eine ausführliche Javadoc[32]-Spezifikation des Frameworks und ein Leitfaden, der verschiedene Aspekte des Frameworks beschreibt. Darüber hinaus gibt es einige Beispiele im Inter-

[32]Javadoc ist ein Tool, das es ermöglicht, aus Kommentaren innerhalb von Java-Code eine Dokumentation im HTML-Format zu erzeugen (siehe [Orae]).

net und ein Forum für die Verwendung des JGraph-Frameworks (vgl. [JGra]). Da das JGraphX-Framework eine Neuentwicklung des JGraph-Frameworks darstellt, können viele Beispiele für die Verwendung des JGraphX-Frameworks übernommen werden.

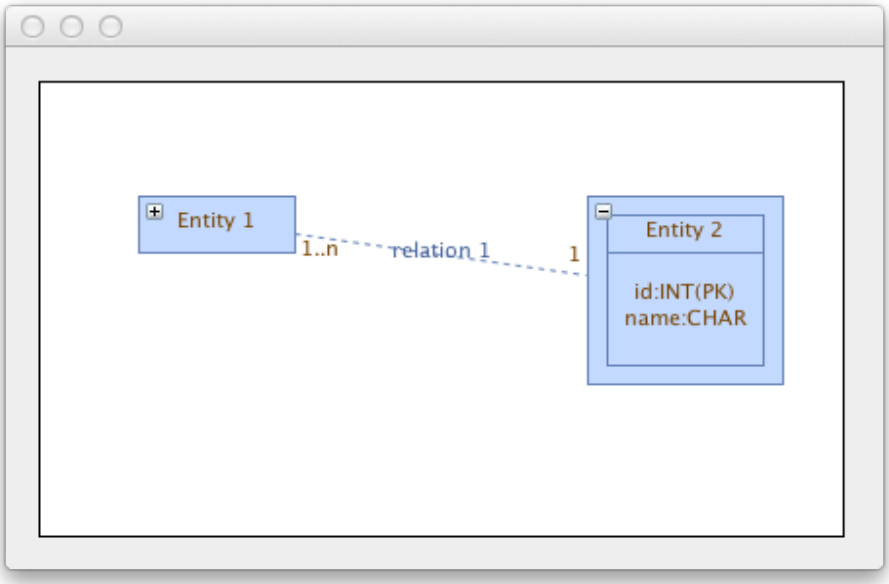


Abbildung 12: *Beispielanwendung zeigt die angepasste Visualisierung eines Graphen mit dem JGraphX-Framework.*

Abbildung 12 zeigt, dass die Visualisierung eines Graphen mit dem JGraphX-Framework detailliert ist und relativ einfach angepasst werden kann. Knoten können als Entität und Kanten können gestrichelt oder durchgehend mit der jeweiligen Kardinalität und Bezeichnung als Text visualisiert werden. Die Möglichkeit, Kanten entsprechend der Krähenfuß-Notation zu visualisieren, wurde noch nicht explizit evaluiert. Allerdings kann gesagt werden, dass JGraphX das Importieren und Verwenden von sogenannten Dia-Shape-Dateien (siehe Benutzerhandbuch, vgl. [JGrc]) ermöglicht. Mithilfe dieser Dateien können eigene Formen für Knoten oder Kanten verwendet werden, was eine Möglichkeit sein könnte, die Krähenfuß-Notation für die Darstellung der Kanten zu verwenden. Bezüglich der Interaktion ist es möglich, Knoten zu bewegen, wobei Kanten erhalten bleiben. Auch Aktionen, die im Zusammenhang mit Mausklicks stattfinden, können angepasst und erweitert

werden.

4.3.3 yFiles

Bei dem yFiles-Framework handelt es um ein Java-Framework, das „Algorithmen und Komponenten für die Analyse, die Visualisierung und das automatische Anordnen von Graphen, Diagrammen und Netzwerken zur Verfügung stellt“ (Bezug zu [yWo]).

Da es sich bei dem yFiles-Framework um ein kommerziell vertriebenes Framework und kein Open-Source-Framework konnte die Verwendung dieses Frameworks für den Einsatz im Rahmen dieser Arbeit nicht evaluiert werden. Da es aber ziemlich mächtig zu sein scheint und die geforderten Anforderungen bezüglich der visuellen Darstellung erfüllt, soll es hier dennoch kurz aufgeführt werden.

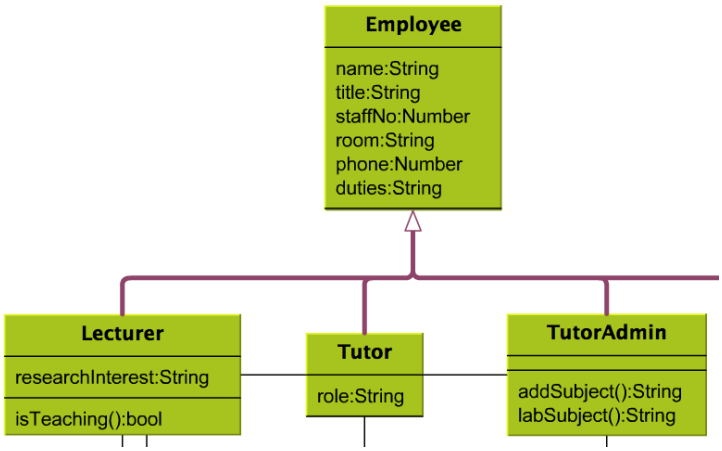


Abbildung 13: *Beispiel für eine Anwendung, die mithilfe des yFiles-Frameworks entwickelt wurde.*

Laut Angaben der Entwickler soll die Verwendung des Frameworks es beispielsweise ermöglichen, Anwendungen für das Modellieren verschiedener Diagramme, wie UML- oder Flowchart-Diagramme, zu implementieren. Zwei Beispiele dazu werden in Abbildung 13 und 14 gezeigt.

Da das Framework es erlaubt, die grafische Darstellung eines Graphen sehr detailliert anzupassen, bzw. es sogar explizit für die Implementierung einer Anwendung zum Zeichnen verschiedener Modelle vorgesehen ist, könnte es im Rahmen dieser Arbeit prinzipiell verwendet werden. Nicht zuletzt,

weil davon ausgegangen werden kann, dass das Framework ausführlich dokumentiert wurde und Hilfestellung geboten wird, das es kostenpflichtig ist. Diese Tatsache ist hierbei Segen und Fluch zugleich, da im Rahmen dieser Arbeit kein Budget zu Verfügung steht, und das Framework somit bedauerlicherweise nicht verwendet werden kann.

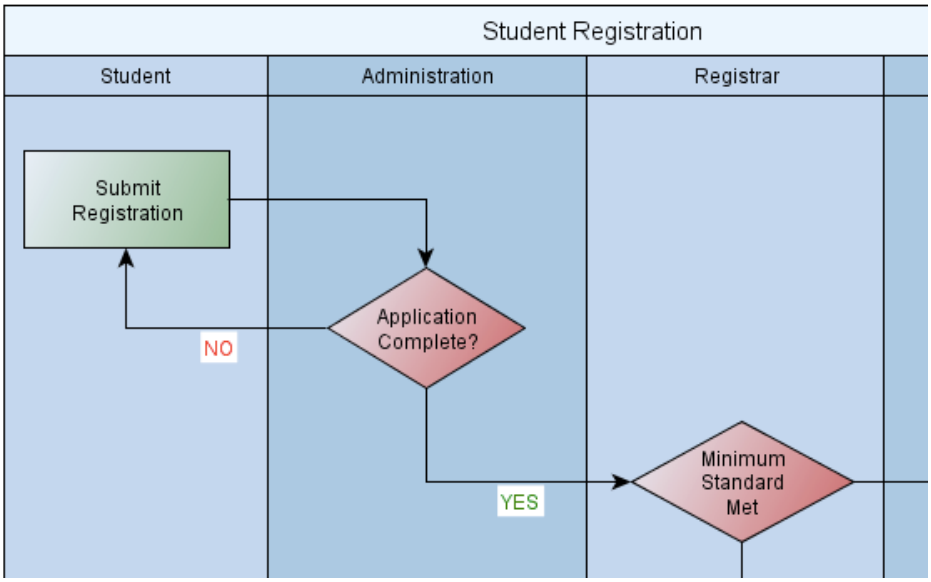


Abbildung 14: *Beispiel für eine Anwendung, die ebenfalls mithilfe des yFiles-Frameworks entwickelt wurde.*

4.4 Fazit

Der Vergleich durch die Implementierung kleinerer Beispielanwendungen mithilfe der aufgeführten Frameworks hat gezeigt, dass sich JGraphX-Framework am besten eignet. Das Implementieren der Beispielanwendung (siehe Abbildungen 11 und 12) mit dem JGraphX-Framework hat sich sehr einfach gestaltet und nicht mehr als 60 Zeilen Code benötigt. Die verschiedenen Anpassungen der Darstellung, entsprechend den primären Anforderungen, konnten dabei mit sehr geringem Aufwand realisiert werden. In den meisten Fällen war lediglich das Ändern von Parametern nötig.

Weiterhin übernimmt JGraphX viele Aspekte der Visualisierung von Graphen innerhalb eines JPanels, wie das Bewegen und Markieren von Elemen-

ten. Durch diese Tatsache kann noch mehr an Entwicklungsaufwand eingespart werden, da die Anforderung F30 beispielsweise nicht explizit implementiert werden muss.

5 Entwurf

Die letzten Abschnitte haben gezeigt, welche Anforderungen an die zu entwickelnde Anwendung gestellt werden und welches Framework sich für die Visualisierung des Entity-Relationship-Diagramms eignet. In diesem Abschnitt werden Entwürfe vorgestellt, die sich aus den Anforderungen ergeben und als Basis für die Architektur der Anwendung dienen. Die Entwürfe werden dabei in Form von verschiedenen Diagrammen gezeigt.

5.1 Anwendungsfälle

Ein sogenanntes Anwendungsfalldiagramm entsprechend der UML-Notation zeigt, welche Anwendungsfälle von einem Aktor[33] mithilfe der zu entwickelnden Anwendung durchgeführt werden können. Das folgende Anwendungsfalldiagramm zeigt eine Übersicht der Anwendungsfälle, welche direkt den primären Anforderungen aus Abschnitt 3.1.1 zugeordnet werden können (siehe Abbildung 15).

[33]Ein Aktor, in Bezug auf ein Anwendungsfalldiagramm, ist eine Ausführende Instanz, meistens handelt es sich um den Anwender. Ein System kann allerdings auch ein Aktor sein.

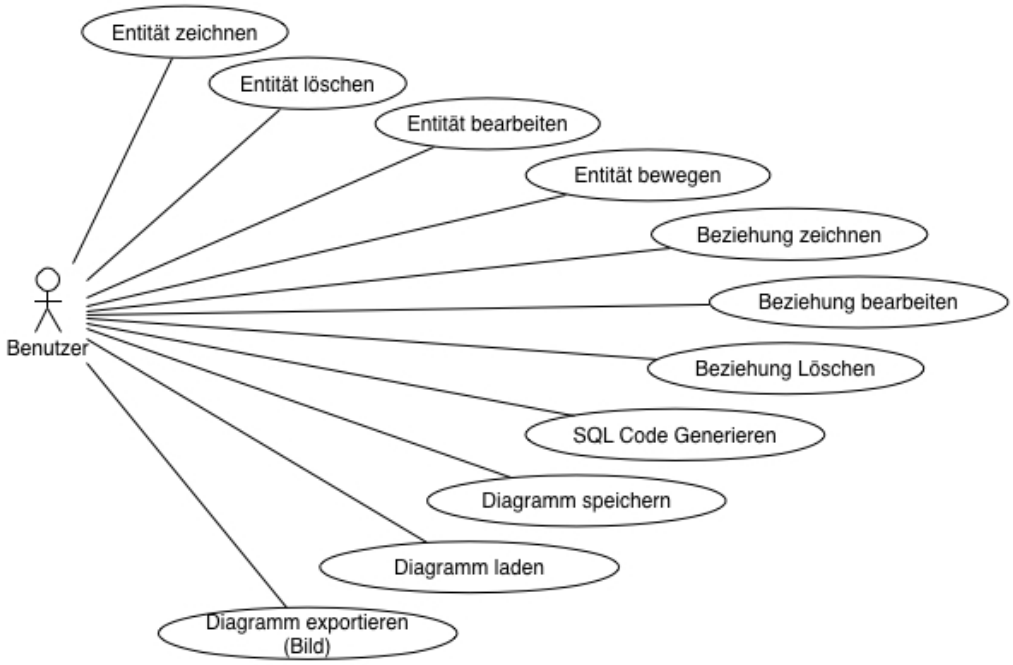


Abbildung 15: *Anwendungsfalldiagramm resultiert aus primären, funktionalen Anforderungen.*

5.2 GUI-Design

Durch das im vorherigen Abschnitt gezeigte Anwendungsfalldiagramm wurde festgelegt, welche Funktionen die zu entwickelnde Anwendung entsprechend den primären Anforderungen anbieten soll. Als nächstes folgt ein Entwurf der grafischen Benutzeroberfläche (GUI).

5.2.1 Hauptfenster

Das Hauptfenster der zu entwickelnden Anwendung ist schlicht gehalten, beinhaltet ein JPanel und links eine Art Leiste mit verschiedenen JButtons[34] (siehe Abbildung 16). Innerhalb des JPanels soll das Entity-Relationship-Diagramm visualisiert werden, und die links befindlichen JButtons erlauben

[34] JButtons sind Benutzerschaltflächen im Rahmen der Programmierung von grafischen Oberflächen mit Java.

die Ausführung verschiedener Funktionen, die im Rahmen des Anwendungsfalldiagramms spezifiziert wurden.

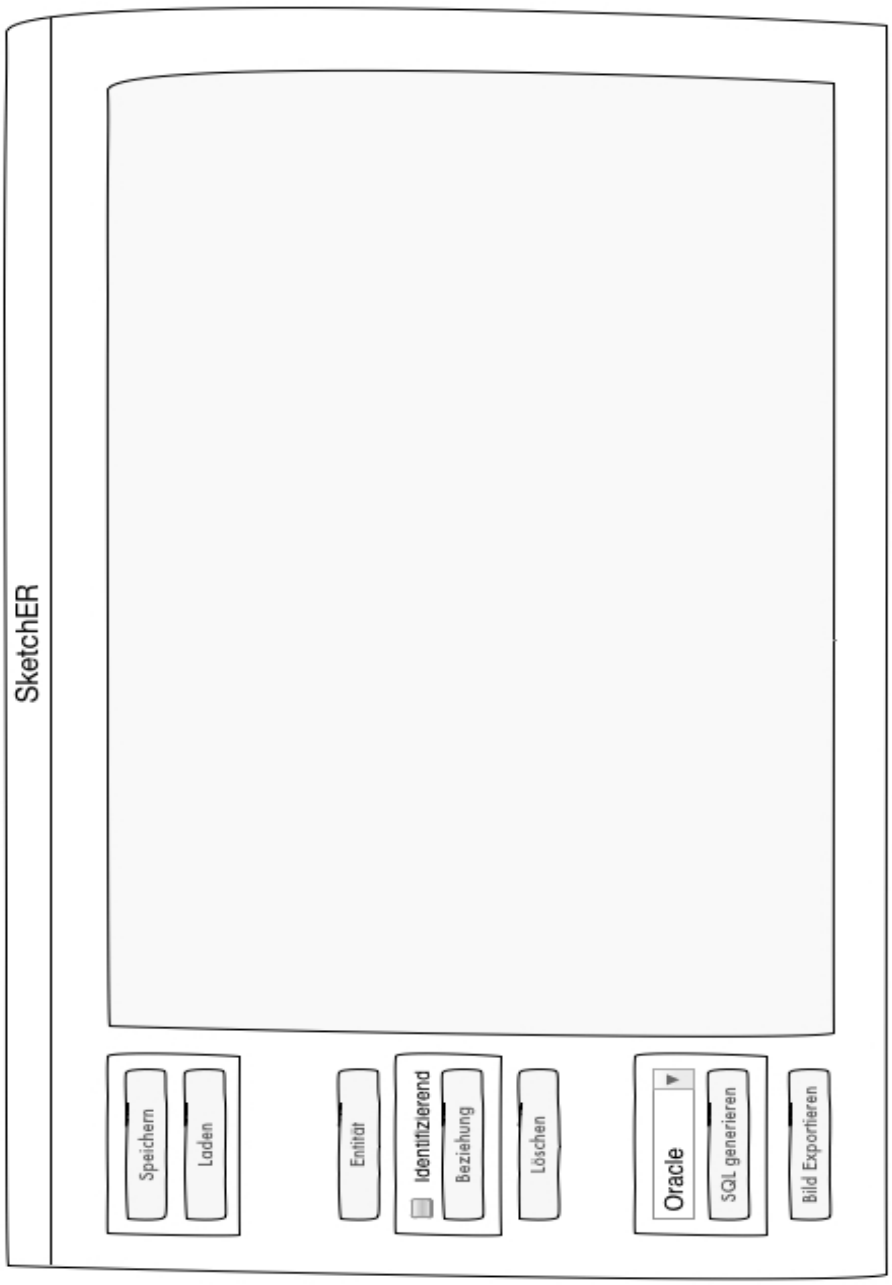


Abbildung 16: *Hauptfenster der zu entwickelnden Anwendung.*

5.2.2 Dialog Entität bearbeiten

Um eine bereits visualisierte Entität zu bearbeiten, wird ein Dialogfenster realisiert. In diesem Dialogfenster können Benutzer die Eigenschaften einer Entität einsehen und bearbeiten (siehe Abbildung 17). Dazu gehört das hinzufügen, löschen und bearbeiten von Attributen, die ihrerseits in einer JTable[35] dargestellt werden.

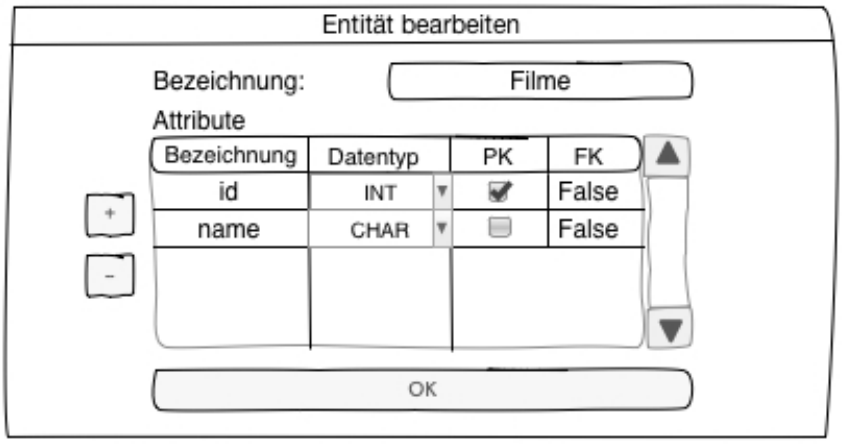


Abbildung 17: *Dialogfenster zum Bearbeiten einer Entität.*

5.2.3 Dialog Beziehung bearbeiten

Um eine bereits visualisierte Beziehung zu bearbeiten, wird ebenfalls ein Dialogfenster realisiert (siehe Abbildung 18).

[35]Element des Java-Frameworks das für die Darstellung von Tabellen verwendet wird.

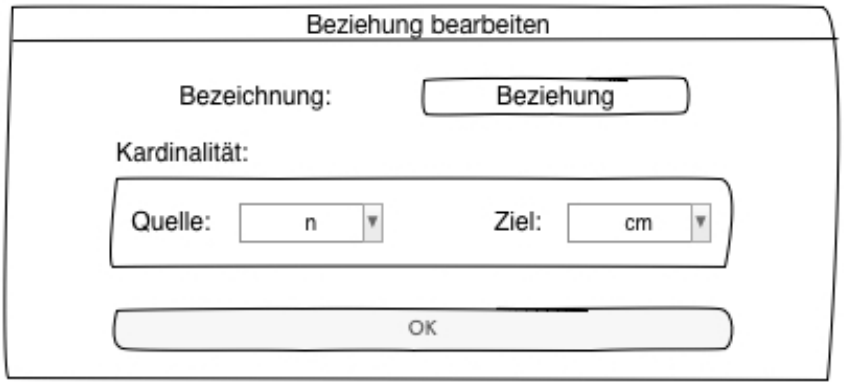


Abbildung 18: *Dialogfenster zum Bearbeiten einer Beziehung.*

5.3 Abläufe

Das Anwendungsfalldiagramm aus Abschnitt 5.1 hat eine Übersicht der Funktionen gezeigt, die durch die zu entwickelnde Anwendung angeboten werden sollen. Dieser Abschnitt zeigt, wie einzelne Abläufe der verschiedenen Anwendungsfälle realisiert werden, sowie das benötigte Statusmodell.

5.3.1 PAP-Diagramme

Die Abläufe der einzelnen Anwendungsfälle werden im Folgenden beschrieben und größtenteils durch sogenannte Programmablaufpläne[36] (kurz PAP) dargestellt:

Entität zeichnen Um eine Entität zu zeichnen, muss der Anwender zunächst auf den Button mit der Beschriftung „Entität“ klicken. Die Anwendung wird dann in den internen Status „Add Entity“ versetzt (das Statusmodell wird im Anschluss genauer erläutert). Nach dem Klick auf den Button muss der Anwender in das JPanel klicken, in dem das Diagramm dargestellt wird. Es wurde bereits angedeutet, dass der Code für die grafische Darstellung und der Code für die Geschäftsobjekte[37] und Geschäftsprozesse voneinander getrennt werden. Falls der interne Status sich in der Zwischenzeit

[36]Ein Programmablaufplan, auch Flussdiagramm genannt, zeigt den Ablauf eines bestimmten Aspekts innerhalb eines zu entwickelnden Programms.

[37]Entitäten und Beziehungen des Entity-Relationship-Diagramms.

nicht geändert hat, wird eine Standard-Entität[38] erzeugt und zunächst einer internen Repräsentation des zu erzeugenden Entity-Relationship-Diagramms hinzugefügt. Bei dem Hinzufügen einer Entität wird zuerst geprüft ob die Entität eingefügt werden kann, oder ob beispielsweise bereits eine Entität mit derselben Bezeichnung existiert. Falls die Entität der internen Repräsentation des Entity-Relationship-Diagramms hinzugefügt wurde, wird die Entität in dem JPanel, an der Position des Mausklicks, visualisiert und der interne Status auf „Neutral“ geändert (siehe Abbildung 19).

[38]Entität mit der Bezeichnung „Entity“ und ohne Attribute.

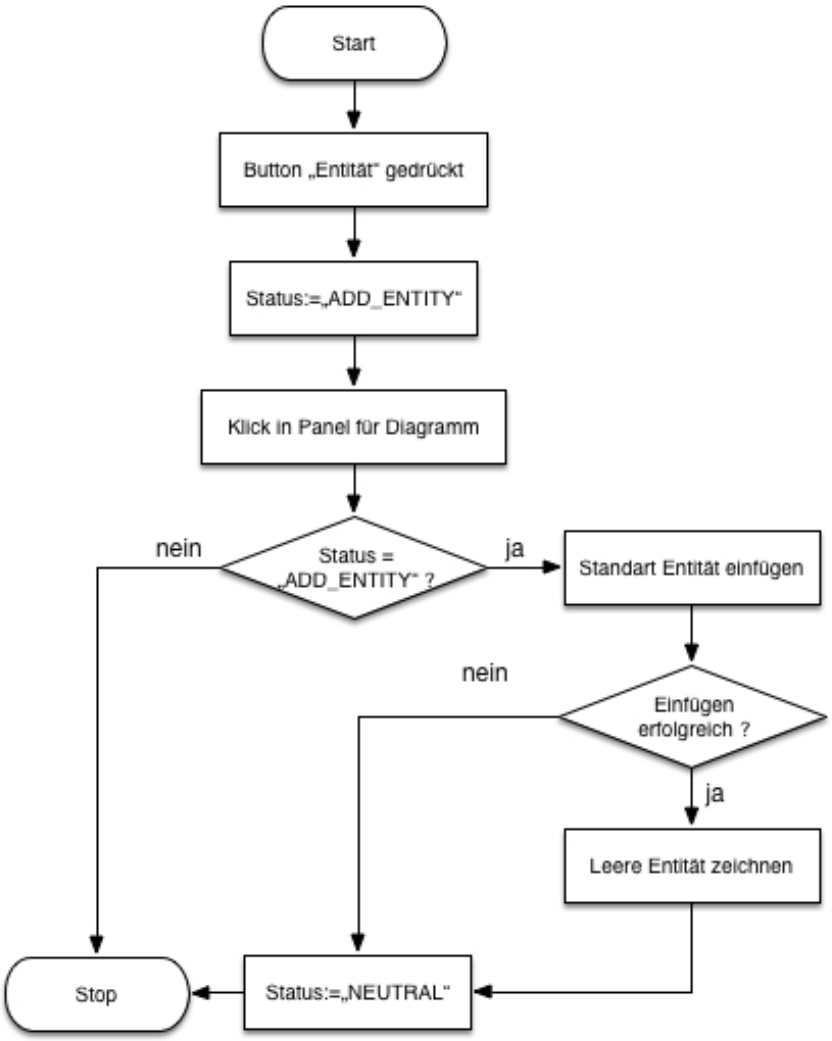


Abbildung 19: *Programmablaufplan für das Hinzufügen einer Entität.*

Entität löschen Um eine Entität zu löschen, muss der Anwender die Entität zunächst innerhalb des JPanels mit einem Mausklick selektieren. Die ALPHA-Version der zu entwickelnden Anwendung unterstützt lediglich das Selektieren eines Elements (Entität oder Beziehung), in Zukunft sollte auch das Selektieren mehrerer Elemente gleichzeitig unterstützt werden. Nachdem eine Entität selektiert wurde, muss der Anwender auf den Button mit der

Beschriftung „Löschen“ drücken.

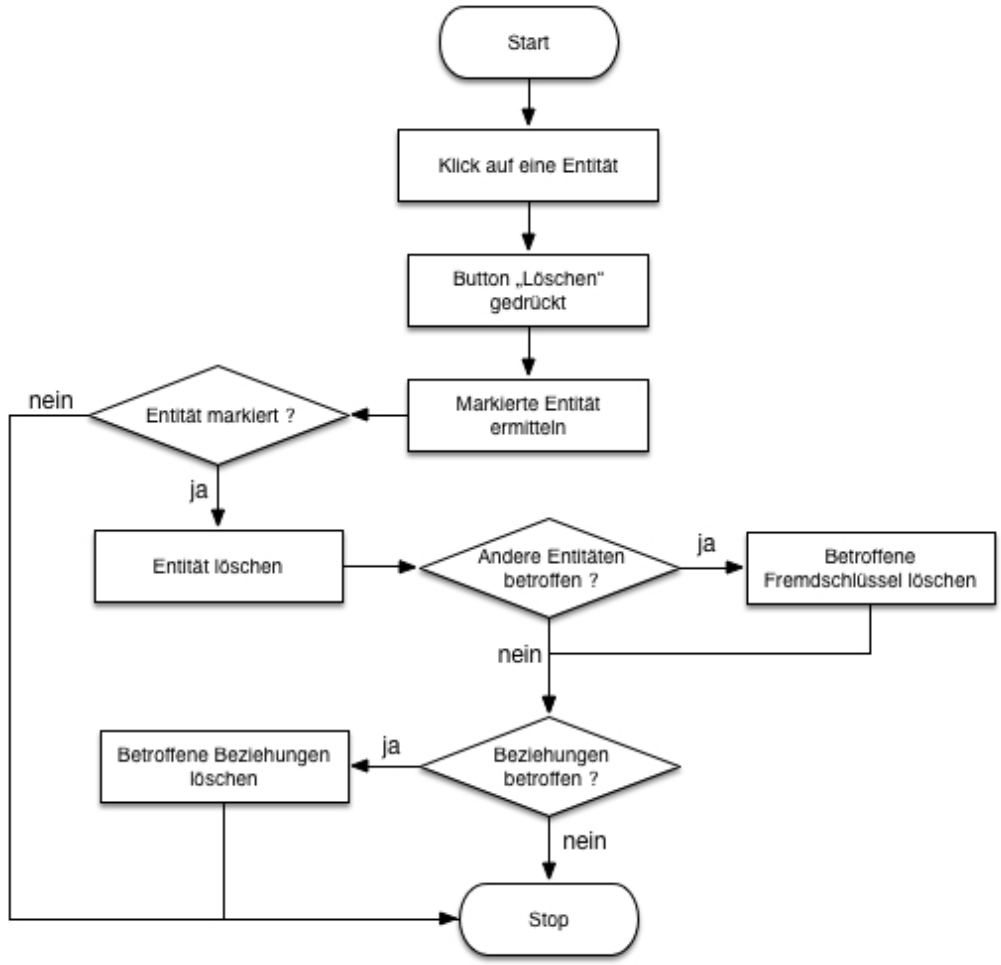


Abbildung 20: *Programmablaufplan für das Löschen einer Entität.*

Nachdem ein Dialog geöffnet wurde, in dem der Anwender bestätigen muss, dass die selektierte Entität gelöscht werden soll, wird ermittelt, welche Entität innerhalb der grafischen Darstellung markiert wurde. Falls die markierte Entität nicht in der Zeit zwischen dem Markieren und dem Drücken auf den Button mit der Beschriftung „Löschen“ wieder deselektiert wurde, wird die Entität aus der internen Repräsentation und der grafischen Darstellung des Entity-Relationship-Diagramms entfernt. Daraufhin wird geprüft,

ob andere Entitäten durch Fremdschlüsseleinträge und Beziehungen von dem Löschen der Entität betroffen sind. Fremdschlüsseleinträge der betroffenen Entitäten werden aus der internen Repräsentation des Entity-Relationship-Diagramms gelöscht und die grafische Darstellung der betroffenen Entitäten aktualisiert (siehe Abbildung 20). Betroffene Beziehungen werden ebenfalls aus der internen Repräsentation des Entity-Relationship-Diagramms gelöscht, allerdings muss die grafische Darstellung hier nicht aktualisiert werden, da das JGraphX-Framework Kanten zu einem gelöschten Knoten automatisch löscht.

Auf das Abrufen oder Setzen des internen Status kann bei diesem Anwendungsfall verzichtet werden. Der Grund dafür ist, dass das JGraphX-Framework die Selektion von Elementen übernimmt; somit muss dieser Aspekt nicht explizit implementiert werden.

Entität bearbeiten Eine Entität kann bearbeitet werden (siehe Abbildung 21), indem der Anwender einen Doppelklick auf einer zu bearbeitenden Entität innerhalb des Panels durchführt. Anschließend wird ermittelt, auf welcher Entität innerhalb der grafischen Darstellung der Doppelklick durchgeführt wurde, und der Dialog zum Bearbeiten einer Entität (siehe Abbildung 17) gestartet.

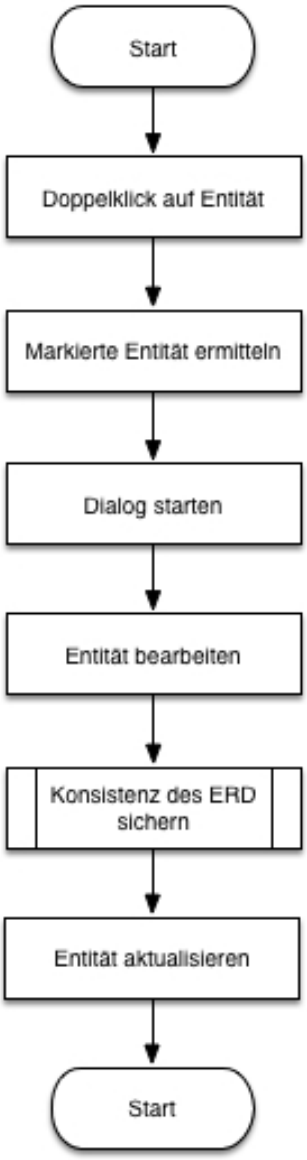


Abbildung 21: *Programmablaufplan für das Bearbeiten einer Entität.*

Innerhalb des Dialogs können Attribute hinzugefügt, gelöscht sowie deren Bezeichnung, Datentyp und die Primärschlüsseleigenschaft verändert werden. Handelt es sich um ein Fremdschlüsselattribut, kann dieses nicht gelöscht

werden und der Datentyp kann nicht verändert werden.

Nachdem der Anwender die Entität bearbeitet hat, werden die Änderungen in die interne Repräsentation des Entity-Relationship-Diagramms übernommen, wobei ein ausgefeilter Algorithmus (siehe Abbildung 22) für die Konsistenz sorgt. Der Algorithmus prüft im Wesentlichen, ob ein Primärschlüsselattribut gelöscht wurde oder einem Attribut die Primärschlüsseleigenschaft entzogen wurde. Ist dies der Fall, wird ein Dialog gezeigt, mit der Warnung, dass das Löschen eines Primärschlüsselattributs auch das Löschen von Fremdschlüsseln, die das Primärschlüsseattribut referenzieren, nach sich ziehen kann. Bestätigt der Anwender, so werden die entsprechenden Fremdschlüsseleinträge entfernt.

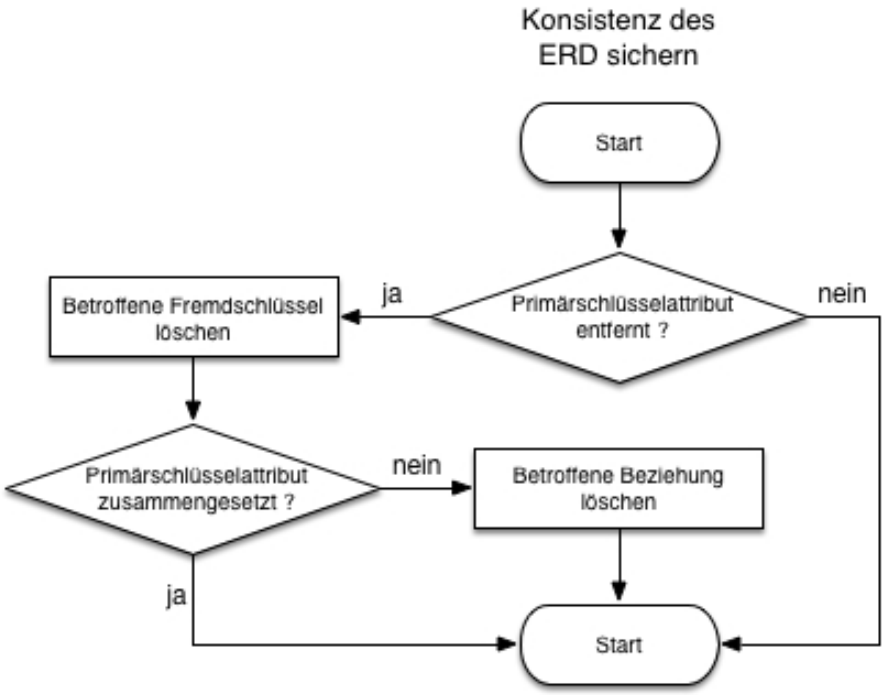


Abbildung 22: *Programmablaufplan für das Bearbeiten einer Entität.*

Handelt es sich bei dem Primärschlüssel der Quell-Entität um ein einzelnes Attribut, also einen **nicht** zusammengesetzten Schlüssel, so wird die jeweilige Beziehung ebenfalls gelöscht. Ist der Primärschlüssel zusammengesetzt, werden nur die entsprechenden Fremdschlüsseleinträge gelöscht.

Nachdem die Konsistenz der internen Repräsentation des Entity-Relationship-Diagramms gesichert wurde, werden die Entität sowie eventuell durch das Löschen von Fremdschlüsseleinträgen betroffene andere Entitäten und Beziehungen innerhalb der grafischen Darstellung aktualisiert.

Beziehung bewegen Das Bewegen von Entitäten innerhalb des JPanels muss nicht explizit implementiert werden, da dieser Aspekt bereits – wie das Selektieren von Elementen – innerhalb des JGraphX-Frameworks implementiert ist.

Beziehung zeichnen Eine Beziehung zwischen zwei Entitäten kann erzeugt werden, indem der Anwender den Button mit der Beschriftung „Beziehung“ drückt, woraufhin die Anwendung in den internen Status „Add Relation“ versetzt wird. Anschließend muss der Anwender auf die Quell-Entität der zu erzeugenden Beziehung innerhalb des JPanels klicken. Falls der Klick keine Entität „getroffen“ hat, wird der Vorgang abgebrochen und die Anwendung wieder in den internen Zustand „Neutral“ versetzt. Wurde allerdings eine Entität „getroffen“, so wird diese selektiert und zunächst als Quell-Entität der zu erzeugenden Beziehung intern gespeichert. Parallel dazu wird die Anwendung in den internen Zustand „First Participant Selected“ versetzt.

Nach dem Klick auf die Quell-Entität, muss der Anwender auf die Ziel-Entität klicken. Dazu wird der Ablauf ab dem Klicken einer Entität erneut ausgeführt. Falls der interne Status sich in der Zwischenzeit nicht geändert hat, wird eine Standard-Beziehung (1-n-Beziehung) von der Quell- zu der Ziel-eEntität erzeugt. (Falls sich der Status doch in der Zwischenzeit geändert hat, wird der Vorgang ebenfalls abgebrochen und die Anwendung wieder in den internen Status „Neutral“ versetzt.) Diese erzeugte Beziehung wird der internen Repräsentation des Entity-Relationship-Diagramms hinzugefügt, wobei auch der/die entsprechende(n) Fremdschlüssel der Ziel-Entität hinzugefügt wird/werden. Im Anschluss wird die Beziehung visualisiert und die Darstellung der betroffenen Ziel-Entität aktualisiert und die Anwendung wieder in den Status „Neutral“ versetzt (siehe Abbildung 23).

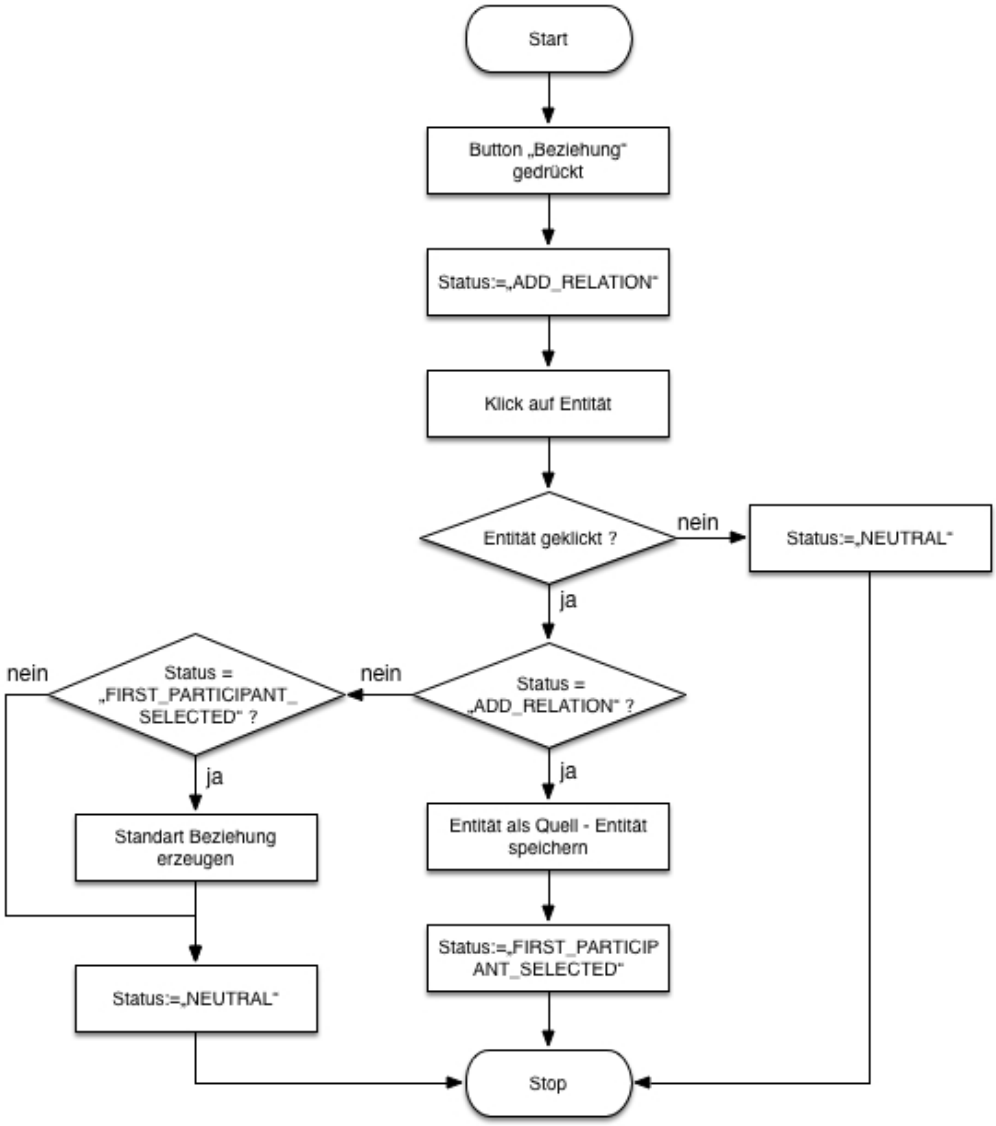


Abbildung 23: *Programmablaufplan für das Hinzufügen einer Beziehung zwischen zwei Entitäten.*

Beziehung bearbeiten Für das Bearbeiten einer bereits visualisierten Beziehung zwischen zwei Entitäten wird, wie für das Bearbeiten einer Entität, ein Dialog verwendet. Dieser wird durch einen Doppelklick auf die zu bearbei-

tende Beziehung gestartet und mit den Eigenschaften der Beziehung gefüllt. Bei dem Befüllen der GUI-Elemente, wird die Auswahl der Kardinalitäten so eingeschränkt, dass die Beziehung nicht „umgedreht“ werden kann. Beispielsweise kann eine 1-n-Beziehung, nicht in eine n-1-Beziehung innerhalb des Dialogs umgewandelt werden. Für eine zu bearbeitende 1-n-Beziehung kann für die Kardinalität der Quell-Entität, lediglich 1 oder C gewählt werden. Für die Ziel-Entität kann in diesem Fall nur 1, C, N oder CN gewählt werden.

Es soll hier auch gesagt sein, dass n-m-Beziehungen nicht explizit angeboten werden. Um eine n-m-Beziehung zu zeichnen, muss der Anwender selber die Zwischen-Entität zeichnen und dann die beiden 1-n-Beziehungen zu der Zwischen-Entität erzeugen.

Nachdem die Beziehung innerhalb des Dialogs von dem Anwender bearbeitet wurde, wird die interne Repräsentation und die grafische Darstellung des Entity-Relationship-Diagramms aktualisiert (siehe Abbildung 24).

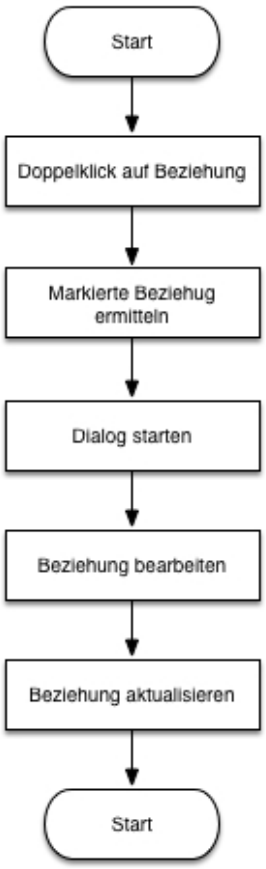


Abbildung 24: *Programmablaufplan für das Bearbeiten einer bereits visualisierten Beziehung zwischen zwei Entitäten.*

Beziehung löschen Um eine Beziehung zu löschen, muss der Anwender zunächst die zu löschende Beziehung innerhalb des JPanels mit einem Mausklick selektieren. Nachdem die zu löschende Beziehung selektiert wurde, muss der Anwender auf den Button mit der Aufschrift „Löschen“ drücken. Anschließend wird ermittelt, welche Beziehung innerhalb der grafischen Darstellung selektiert wurde. Falls die selektierte Beziehung in der Zeit zwischen dem Selektieren und dem Drücken **nicht** wieder deselektiert wurde, wird die Beziehung aus der internen Repräsentation und der grafischen Darstellung des Entity-Relationship-Diagramms entfernt. Weiterhin werden die entsprechenden Fremdschlüsseleinträge aus der Ziel-Entität entfernt und die grafi-

sche Darstellung der Ziel-Entität aktualisiert (siehe Abbildung 25).

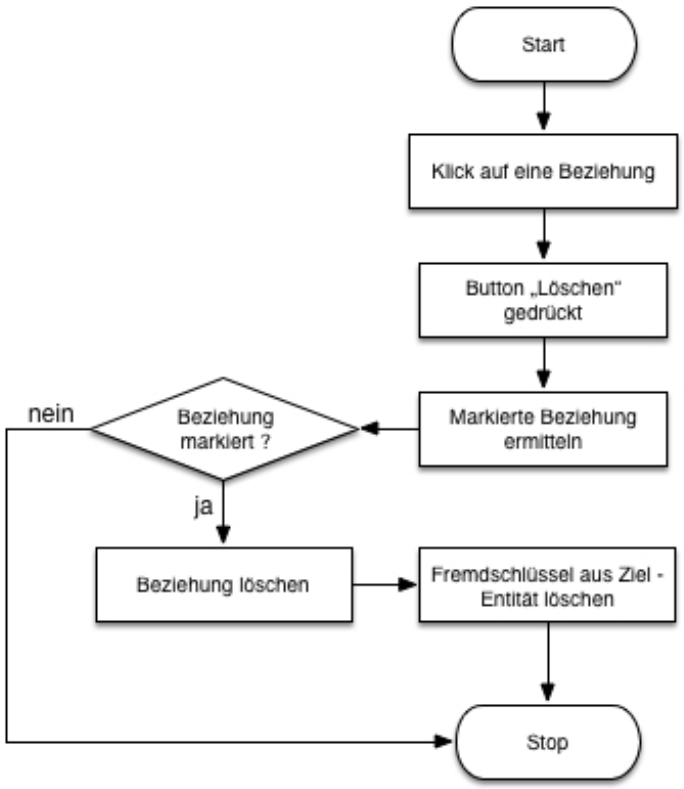


Abbildung 25: *Programmablaufplan für das Löschen einer bereits visualisierten Beziehung zwischen zwei Entitäten.*

SQL-Code generieren Um aus einem gezeichneten Entity-Relationship-Diagramm entsprechenden SQL-Code zu generieren, muss der Anwender zunächst den Dialekt des zu erzeugenden SQL-Code in der entsprechenden JCombobox[39] auswählen und auf den Button mit Aufschrift „SQL generieren“ drücken. Im Anschluss wird ein sogenannter Save-File-Dialog[40] gestartet, in dem der Anwender einen Pfad für die zu speichernde Datei auswählen

[39]Eine JCombobox ist die Auswahlliste des Java-Frameworks, auch Drop-Down-Menü genannt.

[40]vgl. [Orad]

kann. Nachdem der entsprechende Pfad ausgewählt wurde, wird der SQL-DDL-Code erzeugt, indem über alle Entitäten innerhalb der internen Repräsentation des Entity-Relationship-Diagramms iteriert wird. Jede Entität innerhalb des Entity-Relationship-Diagramm stellt eine zu erzeugende Tabelle innerhalb des generierten SQL-DDL-Codes dar. Am Ende des Vorgangs befindet sich in der gespeicherten Datei, der entsprechende SQL-Code (mit dem Dialekt einer spezifischen Datenbanktechnologie).

Diagramm speichern Ein gezeichnetes Entity-Relationship-Diagramm kann in eine XML-Datei abgespeichert werden. Dazu muss der Anwender auf den Button mit der Aufschrift „Speichern“ drücken. Anschließend öffnet sich ein sogenannter Save-File-Dialog, in dem der Anwender einen Pfad für die zu speichernde Datei auswählen kann. Nach dem Bestätigen wird das Entity-Relationship-Diagramm in eine XML-Datei in den angegebenen Pfad exportiert.

Diagramm laden Ein bereits abgespeichertes Entity-Relationship-Diagramm kann importiert werden, indem der Anwender auf den Button mit der Aufschrift „Laden“ drückt. Auch hier wird zunächst ein Dialogfenster gestartet (diesmal handelt es sich allerdings um einen sogenannten Open-File-Dialog (vgl. [Orad])), in dem der Anwender – wie gewohnt – die XML-Datei auswählt, aus der die Geschäftsobjekte geladen werden sollen. Anschließend werden die Geschäftsobjekte aus der Datei geladen und innerhalb des JPanels visualisiert.

Diagramm exportieren Um ein gezeichnetes Entity-Relationship-Diagramm in Form eines Bildes zu exportieren, muss der Anwender auf den Button mit der Aufschrift „Bild exportieren“ drücken. Daraufhin öffnet sich wieder ein Save-File-Dialog, in dem der Anwender den Pfad und den Datentyp des zu exportierenden Bildes festlegen kann. Nach der Bestätigung des Dialogs wird ein Bild an dem festlegten Pfad erzeugt, das den gesamten Inhalt des JPanels beinhaltet. Das Erzeugen des Bildes wird hierbei ohne ein externes Framework realisiert, sondern mit java-internen Mitteln.

5.3.2 Statusmodell

Der vorherige Abschnitt hat die Abläufe der verschiedenen Anwendungsfälle gezeigt, wobei oftmals der interne Status der Anwendung einbezogen wur-

de. Die Eigenschaft der Anwendung, sich in verschiedenen Status zu befinden, ist aus folgendem Grund notwendig: Je nach Anwendungsfall werden verschiedene Aktionen mit einem Mausklick innerhalb des JPanel für die Darstellung des Entity-Relationship-Diagramms verknüpft. Um feststellen zu können, welche Aktion letztlich ausgeführt werden soll, wird daher ein Statusmodell eingeführt. Wie genau die einzelnen Status verwendet werden, um mit einem Mausklick in das JPanels verschiedene Aktionen zu verknüpfen, wurde in dem letzten Abschnitt bereits erklärt. Abbildung 26 zeigt die einzelnen Status, in denen sich die Anwendung befinden kann, und die möglichen Übergänge einzelner Status[41].

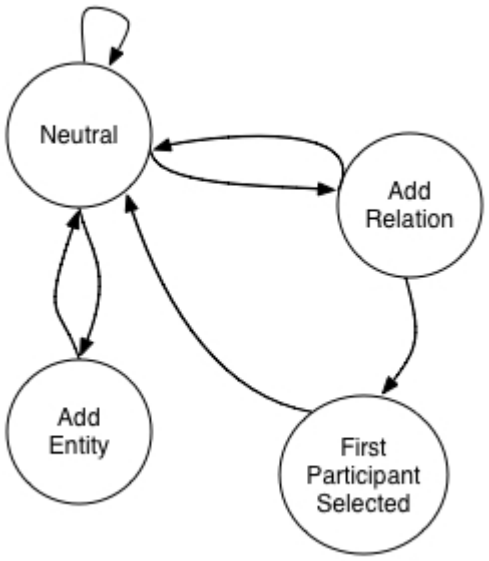


Abbildung 26: *Statusmodell mit einzelnen Status und deren Übergänge.*

[41] Darstellung in Form eines endlichen Automaten (vgl. [FH-]).

6 Architektur

Das letzte Kapitel hat verschiedene Entwürfe gezeigt, die als Basis für die Architektur der zur entwickelnden Anwendung dienen. Die Entwürfe enthalten keine technischen, sondern nur fachliche Aspekte. In diesem Kapitel werden die eher fachlichen Entwürfe mit technischen Aspekten zu einer Softwarearchitektur erweitert.

Die Reihenfolge der Beschreibung einzelner Aspekte der Softwarearchitektur verfolgt den sogenannten Top-down-Ansatz. Das heißt, die Granularität (und somit auch der Abstraktionsgrad) der Sicht auf einen bestimmten Aspekt der Softwarearchitektur wird dabei von Abschnitt zu Abschnitt feiner (von Schichten über Komponenten zu Schnittstellen und Klassen).

6.1 Schichten

Eine sehr grob granuläre Sicht auf die Softwarerchitektur bietet das sogenannte Schichtenmodell (Bezug zu [Dun 9]). Nach den bereits erwähnten Design-Prinzipien „Separation of Concerns“ und „Single Responsibility“ können einzelne zusammenhängende Elemente der Softwarearchitektur in logische Schichten gruppiert oder zusammengefasst werden. Ein wesentlicher Grund für die Aufteilung in Schichten ist, dass die Kommunikation der einzelnen Elemente untereinander minimiert und zentral als Schnittstelle zusammengefasst wird, entsprechend dem Design-Prinzip der losen Kopplung[42]. Die somit lose gekoppelten Schichten können nicht nur leicht ausgetauscht und wiederverwendet werden, einzelne Schichten können sogar auf entfernte System ausgelagert werden als Grundlage für verteilte Anwendungen.

Ein weiterer Aspekt des Schichtenmodells ist die Tatsache, dass eine Schicht jeweils nur auf die unmittelbar darunterliegende Schicht zugreifen kann. Dadurch können beliebig viele Schichten oberhalb einer jeweiligen Schicht „gestapelt“ werden, ohne dass die Schicht selbst davon betroffen ist. Allgemein betrachtet ist eine in logische Schichten unterteilte Softwarearchitektur relativ robust, da Änderungen von jeweiligen Schichten durch die Verwendung von schmalen Schnittstellen kaum Auswirkungen auf andere Schichten haben.

Im Rahmen dieser Arbeit wird die Architektur in eine 3-Schichten-Architektur eingeteilt (siehe Abbildung 27). Bei der untersten Schicht handelt es

[42] vgl. [Dun 9]

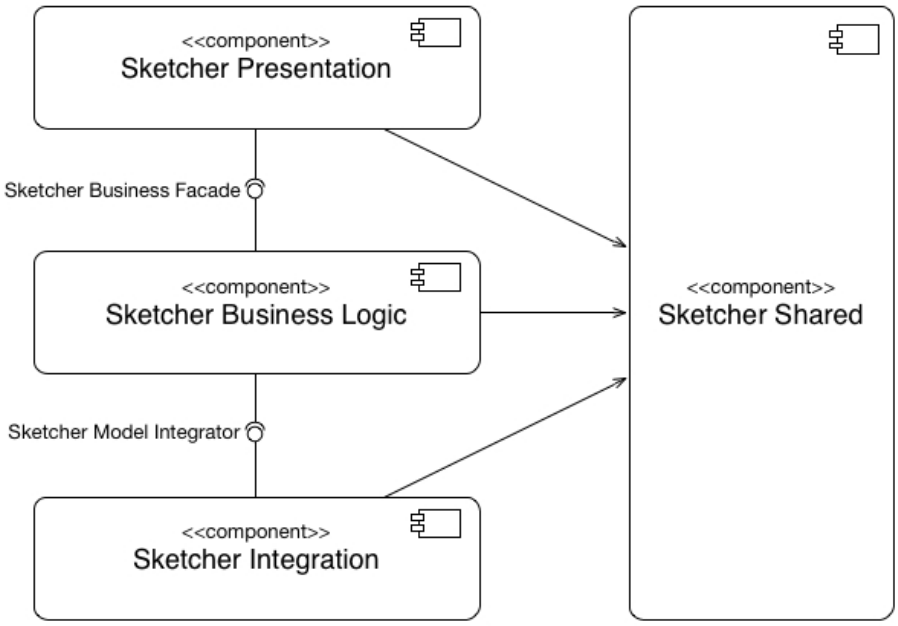


Abbildung 27: *Die 3 Schichten der zu entwickelnden Anwendung.*

sich um die Persistenzschicht, auch Datenhaltungsschicht genannt. Innerhalb der Persistenzschicht wird das Persistieren und Abrufen von Geschäftsobjekten (Elemente des Entity-Relationship-Diagramms) gekapselt. Bei der mittleren Schicht handelt es sich um die Geschäftslogikschicht. Die Geschäftslogikschicht verwendet die Persistenzschicht über eine entsprechende Schnittstelle, um das Persistieren oder Abrufen der Geschäftsobjekten zu realisieren. Darüber hinaus wird in der Geschäftslogikschicht, wie der Name schon andeutet, weitere Logik im Zusammenhang mit dem Umgang von Geschäftsobjekte (Geschäftsprozessen) implementiert und für die Persistenzschicht gekapselt. Die oberste Schicht ist die Präsentations- oder Darstellungsschicht. Innerhalb der Präsentationschicht werden lediglich Elemente implementiert, die für die grafische Benutzeroberfläche zuständig sind. Um Geschäftsprozesse ausführen zu können, verwendet die Präsentationsschicht die Geschäftslogikschicht über die entsprechende Schnittstelle. Die drei Schichten werden um eine "Shared"-Schicht ergänzt, die von allen bereits genannten Schichten verwendet wird.

Es soll an dieser Stelle gesagt sein, dass die Anwendung, die im Rahmen

dieser Arbeit entwickelt wird ein Rich Client[43] sein wird. Die Aufteilung in Schichten dient hier in erster Linie einer übersichtlicheren Gruppierung und robusten Architektur. Darüber hinaus bietet die Aufteilung in Schichten die Möglichkeit, den Rich Client in Zukunft in eine verteilte Anwendung zu transformieren, falls dies irgendwann einmal gewünscht sein sollte.

6.2 Komponenten und Klassen

Der letzte Abschnitt hat eine grobe Übersicht der Softwarearchitektur für die zu entwickelnde Anwendung über das 3-Schichten-Modell gezeigt. In diesem Abschnitt wird die Granularität der Sicht auf die Softwarearchitektur verfeinert. Dazu werden die einzelnen Schichten genauer betrachtet und es wird gezeigt, welche (Sub-)Komponenten sich in den Schichten (Komponenten) verbergen. Für die einzelnen Komponenten werden dann jeweils die enthaltenen Klassen[44] und Schnittstellen gezeigt und beschrieben, was der feinsten Granularität der Sicht auf die Softwarearchitektur entspricht.

6.2.1 Sketcher Shared

Wie bereits erwähnt enthält die „Shared“-Schicht Komponenten, die von allen anderen Schichten genutzt werden. Momentan befindet sich innerhalb der “Shared“-Schicht lediglich die Komponente „Sketcher Model“, was sich in Zukunft aber ändern könnte. Grund dafür ist, dass in Zukunft ggf. neue Funktionen implementiert werden, die ihrerseits für alle oder mehrere Schichten verfügbar sein müssen; das Gleiche gilt für bereits vorhandene Funktionalität. Diese wird dann ebenfalls in die “Shared“-Schicht ausgelagert. Es sie erwähnt, dass der Zugriff auf Elemente dieser Schicht direkt und nicht über eine explizite Schnittstelle der Schicht stattfinden (siehe Abbildung 28).

[43] Alle Schichten befinden sich auf dem Rechner, auf dem die Anwendung ausgeführt wird.

[44] Falls bei Beziehungen innerhalb von Klassendiagrammen, keine Kardinalitä angegeben wurde, handelt es sich um eine 1-1-Beziehung.

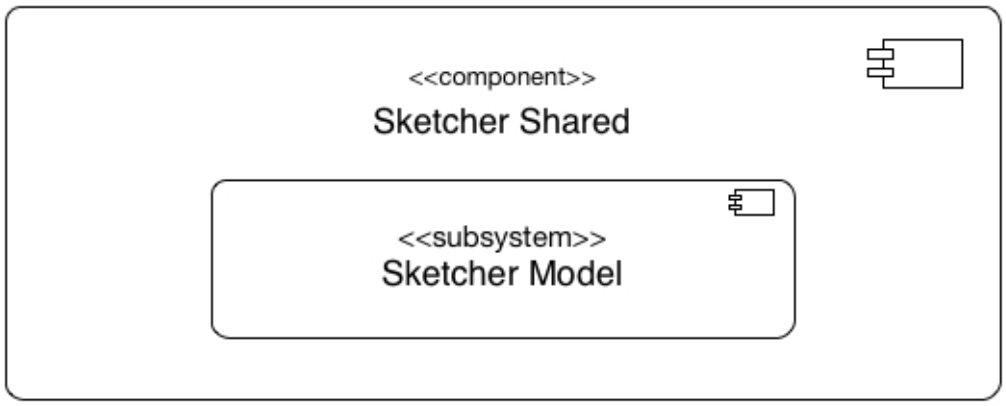


Abbildung 28: *„Shared"-Schicht kann von allen anderen Schichten verwendet werden.*

Sketcher Model Innerhalb dieser Komponente werden die Geschäftsobjekte auf Java-Klassen abgebildet (Geschäftsklassen, siehe Abbildung 29):

SketcherModel Die Klasse „SketcherModel" enthält alle Entitäten und Beziehungen eines Entity-Relationship-Diagramms sowie unter anderem Funktionen für das Hinzufügen oder Löschen von Entitäten und Beziehungen.

SketcherModelRelation Die Klasse „SketcherModelRelation" repräsentiert eine Beziehung und referenziert somit zwei Entitäten und speichert die Kardinalitäten der beiden Entitäten in Bezug auf die Beziehung, sowie die ggf. die Eigenschaft „identifizierend".

SketcherModelRelationCardinality „SketcherModelRelationCardinality" ist eine Java-Enumeration[45] und enthält alle unterstützten Kardinalitäten (1, N, C, CN). Die Enumeration verfügt zusätzlich über eine Methode, die eine gefilterte Auswahl der Kardinalitäten zurückgibt. Die gefilterte Auswahl wird verwendet, um die Auswahlmöglichkeiten der Kardinalität einer Quell-Entität innerhalb des Dialogs zum Bearbeiten einer Beziehung (siehe Abschnitt 5.3.1), bereitstellen zu können.

[45] vgl. [Orac]

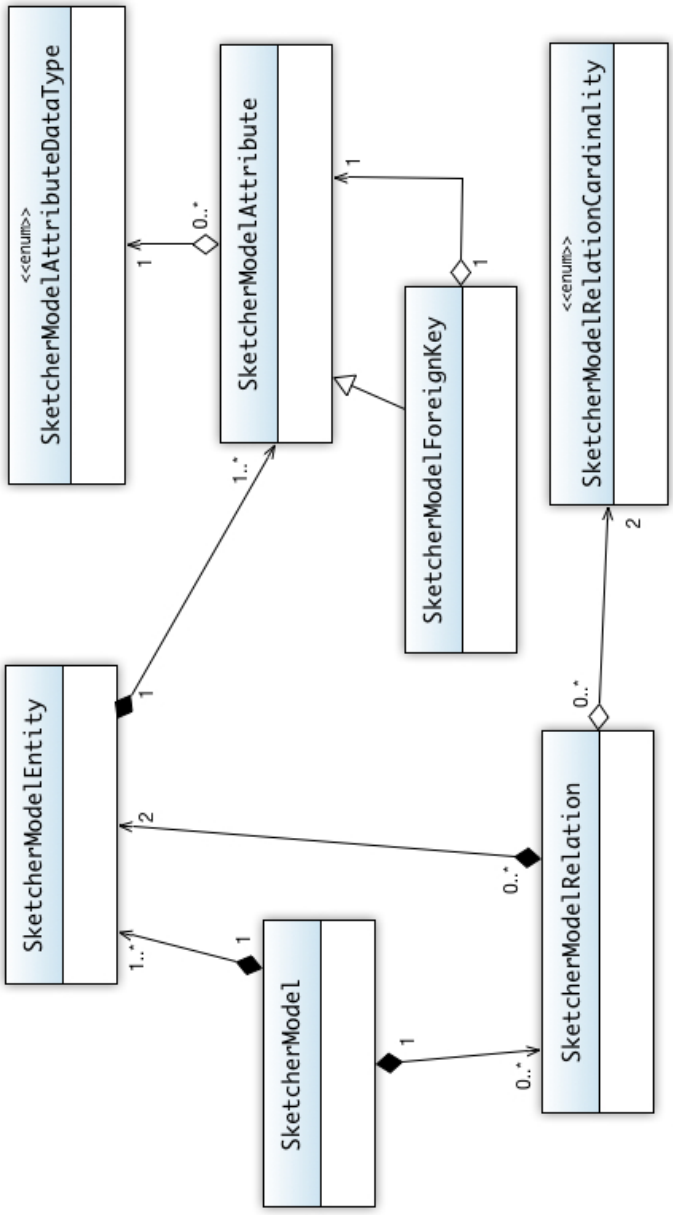


Abbildung 29: *Klassendiagramm zeigt die Geschäftsklassen und deren Beziehung zueinander.*

SketcherModelEntity Diese Klasse stellt eine Entität dar und kann somit verschiedene Attribute aufnehmen. Darüber hinaus wird hier die Bezeichnung einer Entität gespeichert wie auch die Position, an der die Entität innerhalb des JPanels dargestellt werden soll. Des Weiteren verfügt die Klasse über Methoden zum Hinzufügen und Löschen von Attributen und Zurückgeben verschiedener Attributsgruppen (Primärschlüssel, Fremdschlüssel, alle Attribute).

SketcherModelAttribute Die Klasse „SketcherModelAttribute“ speichert die Beschreibung und den Datentyp eines Attributs einer Entität. Darüber hinaus wird über ein entsprechendes Attribut des Typs Boolean[46] festgehalten, ob es sich bei dem Attribut um einen Primärschlüssel handelt oder nicht.

SketcherModelForeignKey Diese Klasse ist eine Erweiterung der Klasse „SketcherModelAttribute“. Da es sich bei einem Fremdschlüsselattribut letztlich auch um ein Attribut einer Entität handelt, enthält ein Fremdschlüsselattribut ebenfalls eine Bezeichnung und einen Datentyp. Zusätzlich dazu wird die Quell-Entität und das von dem Fremdschlüssel referenzierte Primärschlüsselattribut der Quell-Entität gespeichert.

SketcherModelAttributeDatatype Bei diesem Element handelt es sich um eine Java-Enumeration, die alle unterstützen SQL-Datentypen enthält. Die Enumerationen stellen abstrakte SQL-Datentypen dar, die nicht an eine bestimmte Datenbanktechnologie (MySQL, Oracle, DB2[47] etc.) gebunden sind.

[46] Java-Datentyp der einem booleschen Wert entspricht.
[47] Datenbanksystem von IBM, vgl. [IBM]

6.2.2 Sketcher Business Logic

Die “Sketcher Business Logic“-Schicht entspricht der Geschäftslogikschicht des 3-Schichten-Modells und kapselt sämtlichen Zugriff auf Geschäftsobjekte und das Ausführen von Geschäftsprozessen für die Präsentationsschicht. Die Schicht besteht aus den Komponenten „Sketcher Business Facade“ und „Sketcher SQL Generator“ (siehe Abbildung 30).

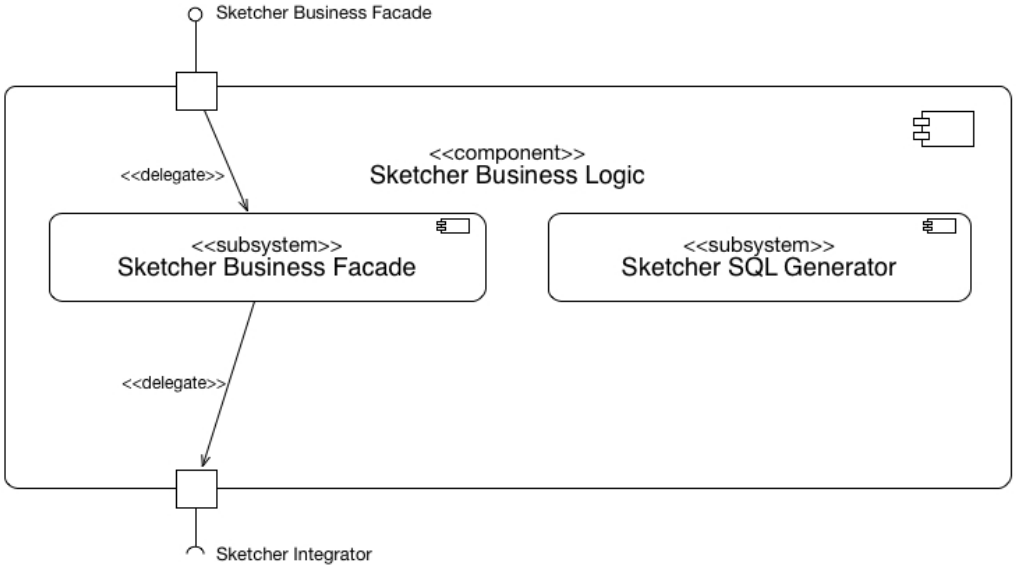


Abbildung 30: *(Sub-)Komponenten und Schnittstellen der “Sketcher Business Logic“-Schicht.*

Sketcher Business Facade Die Komponente „Sketcher Business Facade“ besteht derzeit lediglich aus dem Java-Interface „SketcherBusinessFacade“ und der Klasse „SketcherDefaultBusinessFacade“ und stellt die zentrale Zugriffsstelle für die Präsentationsschicht dar. Dennoch wird die Komponente hier samt Klassendiagramm aufgeführt, um das Zusammenspiel der Komponente mit den Klassen der “Sketcher Modell“-Komponente zu zeigen (siehe Abbildung 31).

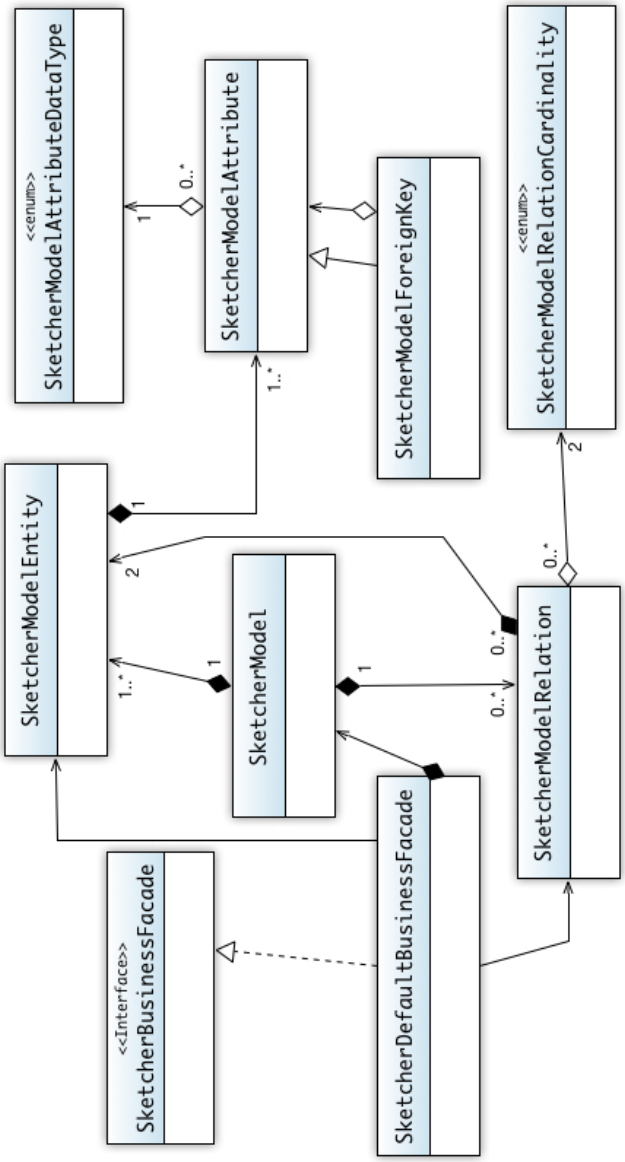


Abbildung 31: *Klassendiagramm zeigt die Interaktion des Java-Interface „SketcherBusinessFacade“ mit den Geschäftsklassen.*

SketcherBusinessFacade Das Java-Interface „SketcherBusinessFacade“ ist die Schnittstelle dieser Schicht, die nach Außen repräsentiert wird.

Die Präsentationsschicht verwendet diese Schnittstelle, um Geschäftsprozesse anzustoßen. Dabei ist es aus Sicht der Präsentationsschicht nicht relevant, welche konkrete Implementierung verwendet wird.

SketcherDefaultBusinessFacade Diese Klasse ist die Standard-Implementierung des im vorherigen Abschnitt genannten Java-Interface „SketcherBusinessFacade“. Innerhalb der Klasse „SketcherBusinessFacade“ werden alle Geschäftsprozesse, zentral implementiert. Die von der Klasse implementierten Methoden lassen sich aus den Anwendungsfällen (vgl. Abschnitt 5.1) ableiten.

Sketcher SQL Generator Die Komponente „Sketcher SQL Generator“ ist für das Generieren von SQL-Code zuständig und besteht derzeit aus der abstrakten Java-Klasse „SketcherSQLGenerator“ und den Java-Klassen „SketcherMySQLGenerator“ und „SketcherOracleGenerator“ (siehe Abbildung 32).

SketcherSQLGenerator Diese abstrakte Klasse ist dafür zuständig, aus den Geschäftsobjekten SQL-Code eines bestimmten Dialekts zu generieren. Da die Datentypen von Attributen innerhalb der Anwendung abstrakt sind, also unabhängig von einer bestimmten Datenbanktechnologie, müssen die abstrakten Datentypen bei dem Generieren von SQL-Code in konkrete Datentypen einer bestimmten Datenbanktechnologie umgewandelt werden. Dieser Aspekt wurde innerhalb dieser Klasse durch eine abstrakte Methode realisiert, damit Unterklassen entscheiden können, wie die abstrakten Datentypen auf konkrete abgebildet werden. Wie genau der SQL-Code generiert wird, wird innerhalb dieser Klasse implementiert, kann aber durch Unterklassen überschrieben werden.

SketcherMySQLGenerator Die Klasse „SketcherMySQLGenerator“ erweitert die im vorherigen Abschnitt eingeführte abstrakte Klasse „SketcherSQLGenerator“ und implementiert die Methode für die Zuordnung der abstrakten SQL-Datentypen zu konkreten MySQL-Datenypen.

SketcherOracleGenerator Diese Klasse erweitert ebenfalls die Klasse „SketcherSQLGenerator“sie implementiert allerdings die Methode für die

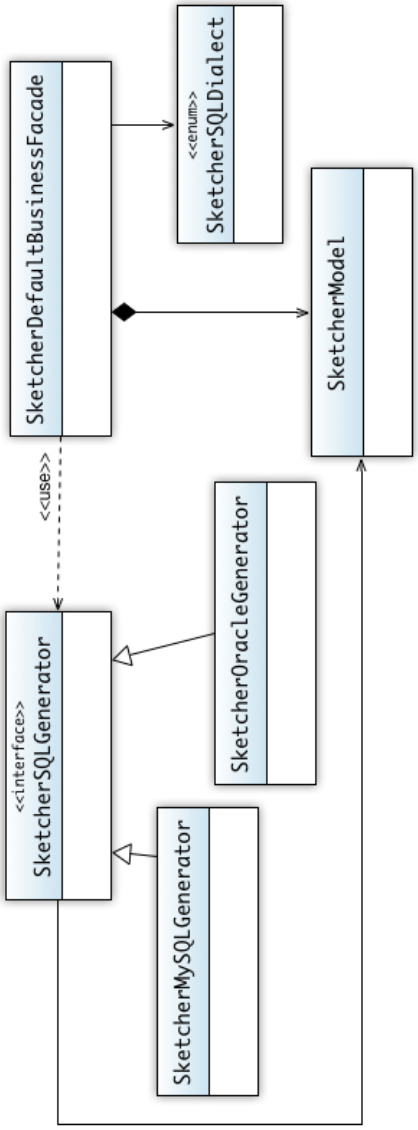


Abbildung 32: *Klassendiagramm zeigt die Java Elemente der Komponente „Sketcher SQL Generator“.*

Zuordnung von abstrakten SQL-Datentypen zu konkreten SQL-Datenbanken der Oracle-Datenbank.

6.2.3 Sketcher Presentation

Die Schicht „Sketcher Presentation“ entspricht der Präsentationsschicht innerhalb des 3-Schichten-Modells und ist lediglich für die grafische Darstellung eines Entity-Relationship-Diagramms zuständig. Dazu enthält die Schicht die Komponenten „Sketcher Main View“, „Sketcher Entity Dialog“ und „Sketcher Relation Dialog“ (siehe Abbildung 33). Für das Anstoßen von Geschäftsprozessen verwendet diese Schicht die Schnittstelle zu der Schicht „Sketcher Business Logic“ („SketcherBusinessFacade“, siehe Abschnitt 6.2.2).

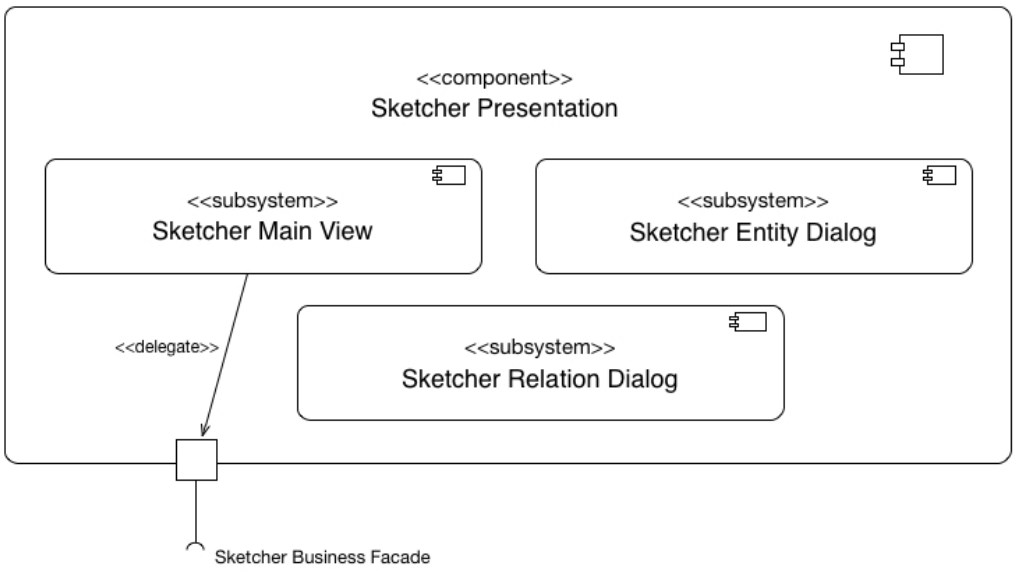


Abbildung 33: *(Sub-)Komponenten der Schicht „Sketcher Presentation“.*

Sketcher Main View Diese Komponente beinhaltet Java-Elemente für die Steuerung des Hauptfensters der Anwendung (siehe Abbildung 34), den Zugriff auf die Schicht „Sketcher Business Logic“ und verwendet die Komponenten „Sketcher Entity Dialog“ und „Sketcher Relation Dialog“.

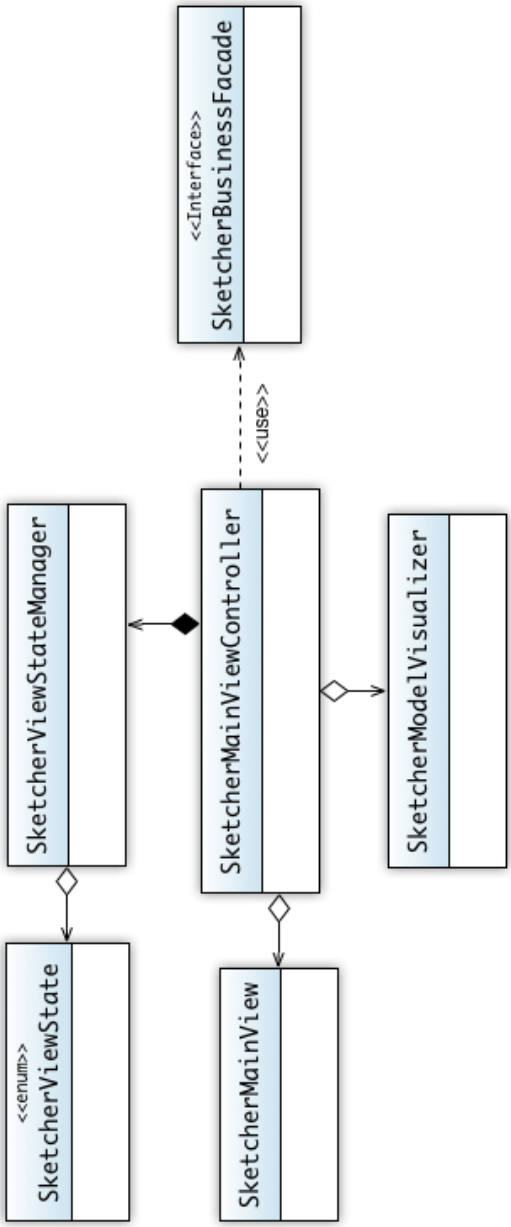


Abbildung 34: *Klassendiagramm zeigt die Elemente für die Benutzerinteraktion innerhalb des Hauptfensters.*

SketcherMainView Die Klasse „SketcherMainView“ repräsentiert das Hauptfenster und beinhaltet alle in Abbildung 16 gezeigten GUI-Elemente. Die Klasse beinhaltet keinerlei fachliche Aspekte und kapselt den Zugriff auf GUI-Elemente über sogenannte Getter-Methoden[48].

SketcherMainViewController Diese Klasse ist das zentrale Element der Komponente „Sketcher Main View“ und steuert die GUI über die Klasse „SketcherMainView“. Genauer gesagt verarbeitet diese Klasse Benutzerinteraktionen und führt Geschäftsprozesse über die Schnittstelle der Schicht „Sketcher Business Logic“ aus. Darüber hinaus verwendet diese Klasse die entsprechenden Elemente der Komponenten „Sketcher Entity Dialog“ und „Sketcher Relation Dialog“, um die Steuerung der Dialogfenster zu realisieren.

Weiterhin ist diese Klasse der indirekte Einstiegspunkt der Anwendung, hier wird die grafische Oberfläche initialisiert und ein Großteil des Programmflusses (gemäß den in Abschnitt 5.3.1 gezeigten Programmablaufplänen) implementiert.

SketcherViewState Bei diesem Element handelt es sich um eine Java-Enumeration, die alle in Abschnitt 5.3.2 gezeigten Status beinhaltet, in denen sich die Anwendung befinden kann.

SketcherViewStateManager Die Klasse „SketcherViewStateManager“ speichert den aktuellen Status, in dem sich die Anwendung befindet, wobei initial der Status „Neutral“ eingenommen wird. Weiterhin bietet die Klasse Methoden für das Ändern und Abfragen des aktuellen Status an.

SketcherModelVisualizer Diese Klasse ist dafür zuständig, Geschäftsobjekte aus der internen Repräsentation des Entity-Relationship-Diagramms zu visualisieren und mit den visualisierten Elementen in Verbindung zu bringen. Man kann die Klasse somit als Schnittstelle oder Vermittler zwischen Geschäftsobjekten und deren visueller Darstellung mithilfe des JGraphX-Frameworks verstehen.

Für die visuelle Darstellung enthält diese Klasse Methoden für das Visualisieren/Entfernen von Entitäten und Beziehungen und die Aktualisierung

[48]Getter-Methoden erlauben den Zugriff auf Instanzvariablen von Java-Klassen mit der Sichtbarkeit „private“ oder „protected“.

deren visueller Darstellung. Innerhalb der Methoden für das Visualisieren von Entitäten und Beziehung wird festgelegt, wie genau diese visuell mithilfe des JGraphX-Frameworks dargestellt werden sollen.

In Abschnitt 4.3.2 wurde erwähnt, dass die Form eines Knotens auf mehreren Wegen verändert werden kann. Die ersten Versuche, die Form eines Knotens so darzustellen, dass die visuelle Darstellung den Anforderungen unter 3.1.1 entspricht, sind allerdings gescheitert.

Grund dafür war ein Denkfehler. Die Form, also die äußere Gestalt der Darstellung **eines** Knotens mithilfe des JGraphX-Frameworks, kann in der Tat auf vielfältige Art und Weise angepasst werden. Allerdings handelt es sich bei der gewünschten Darstellung einer Entität entsprechend Anforderung F10 um **mehrere** Elemente, nämlich ein Rechteck und eine horizontale Linie zum Abtrennen des Kopfbereichs. Mehrere Elemente zu einer einzigen Form zu gruppieren, konnte nicht realisiert werden.

Daher wird der Ansatz gewählt, eine Entität durch die Gruppierung zweier Rechtecke zu realisieren, wobei das obere Rechteck die Bezeichnung und das untere die Attribute der Entität als Aufschrift beinhaltet. Dieser Ansatz lässt sich relativ einfach mithilfe des JGraphX-Frameworks realisieren. So lassen sich mehrere Knoten gruppieren, indem die diese einem Elternknoten zugewiesen werden. Der Elternknoten kann hierbei als Rahmen für die zwei gruppierten Rechtecke gesehen werden, wobei dieser „zusammengeklappt“ werden kann (vgl. Abbildung 12).

Um eine Beziehung entsprechend der Anforderung mit den jeweiligen Kardinalitäten zu visualisieren, wurde ein ähnliches Vorgehen gewählt. Die beiden Kardinalitäten werden in Form von Kindknoten einer Kante hinzugefügt und befinden sich dann auf der jeweiligen Seite der beteiligten Entität.

Um bereits visualisierte Entitäten und Beziehungen zu aktualisieren, weil sie sich innerhalb der internen Repräsentation des Entity-Relationship-Diagramms geändert haben, wird zunächst das entsprechende Objekt innerhalb der internen Repräsentation des JGraphX-Framework ermittelt. Anschließend werden die darzustellenden Aspekte (bei Entitäten die Bezeichnung, bei Attributen die Bezeichnung, der Datentyp und die Schlüsseleigenschaft) an dem verknüpften Geschäftsobjekt und ggf. an dessen visueller Darstellung aktualisiert. Abschließend wird das JPanel, das die visuelle Darstellung des Entity-Relationship-Diagramms enthält aktualisiert, um die Änderungen sofort darzustellen, und nicht erst, wenn das JPanel aufgrund seiner Implementierung entscheidet, „sich“ zu aktualisieren.

Ein weiterer wichtiger Aspekt, der innerhalb dieser Klasse implementiert

wurde und hier nicht unerwähnt bleiben soll, ist das sogenannte „Reshaping“ einer Entität. „Reshaping“ beschreibt die Anpassung der horizontalen und vertikalen Länge (in Pixeln) der visuellen Darstellung einer Entität. Diese Anpassung ist immer dann notwendig, wenn Attribute hinzugefügt oder gelöscht werden und/oder wenn Bezeichnungen innerhalb der visuellen Darstellung länger oder kürzer werden. Innerhalb des „Reshaping“-Prozesses wird für die Anpassung der horizontalen Länge zunächst berechnet, welche Bezeichnung innerhalb der visuellen Darstellung am längsten ist[49]. Die benötigte horizontale Länge einer Entität wird nun berechnet, indem die Anzahl der Zeichen in der längsten Bezeichnung der Entität mit einem konstanten Faktor (momentan 7) multipliziert wird. Um die benötigte vertikale Länge der visuellen Darstellung einer Entität zu berrechnen, wird zunächst die Anzahl der Attribute mit dem konstanten Faktor 15 multipliziert. Anschließend wird die konstante vertikale Länge (10) des Kopfbereichs der visuellen Darstellung der Entität addiert.

Es wurde bereits erwähnt, dass die visuelle Darstellung einer Entität mithilfe des JGrahpX-Frameworks, in Form eines Elternknotens mit zwei Kindknoten realisiert wurde. Dabei stellt ein Kindknoten den Kopfbereich, der andere den Bereich für die Darstellung der Attribute innerhalb einer Entität dar. Falls die horizontale und/oder vertikale Länge einer Entität (Elternknoten) im Rahmen des „Reshaping“-Prozesses angepasst werden muss, werden die Längen der Kindknoten auch entsprechend neu berechnet.

Sketcher Entity Dialog Diese Komponente beinhaltet Java-Klassen für das Bearbeiten einer Entität und die Steuerung des entsprechenden Dialogfensters (siehe Abbildung 35).

SketcherEntityDialog Die Klasse „SketcherEntityDialog“ stellt das Dialogfenster zum Bearbeiten einer Entität dar und beinhaltet alle in Abbildung 17 gezeigten GUI-Elemente. Diese Klasse beinhaltet ebenfalls keine fachlichen Aspekte, der Zugriff auf Elemente der GUI werden über Getter-Methoden gekapselt.

[49]Dabei wird die Bezeichnung der Entität und der Attribute einbezogen. Bei den Attributen wird die Bezeichnung des Attributs, der jeweilige Datentyp und die Schlüsseleigenschaft berücksichtigt.

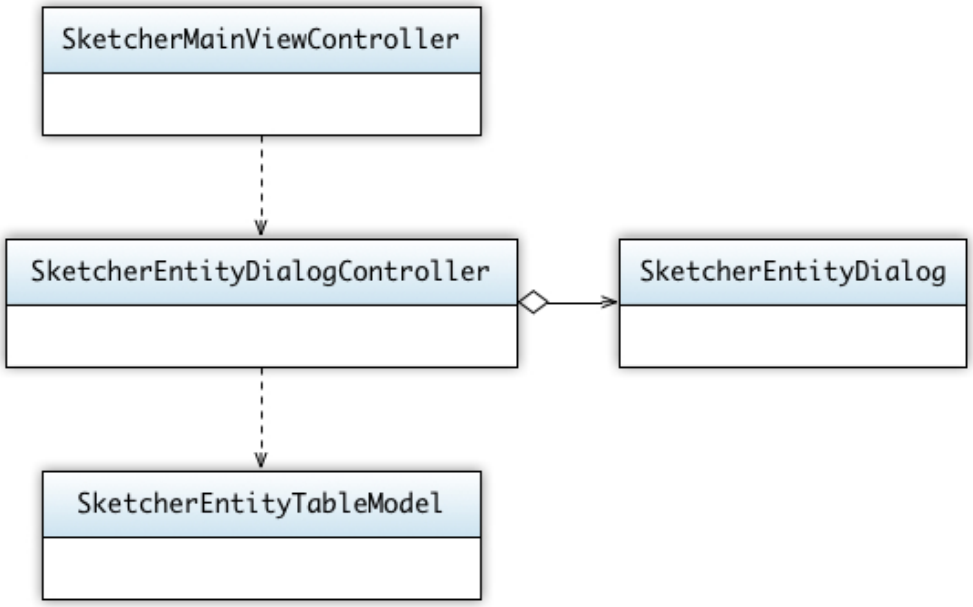


Abbildung 35: *Klassendiagramm zeigt die Elemente für die Benutzerinteraktion innerhalb des Dialogfensters zum Bearbeiten einer Entität.*

SketcherEntityDialogController Diese Klasse steuert die GUI des Dialogfenster über die Klasse „SketcherEntityDialog“. Benutzerinteraktionen werden hier zwar verarbeitet, allerdings hat diese Klasse nur eingeschränkten Zugriff auf das Ausführen von Geschäftsprozessen. Der Grund dafür ist, dass die zu bearbeitende Entität dem Dialogfenster als Referenz übergeben wird. Innerhalb dieser Klasse wird lediglich die Referenz bearbeitet; die Integrität der internen Repräsentation des Entity-Relationship-Diagramms aufrecht zu erhalten, ist hauptsächlich Aufgabe der Klasse „SketcherMainViewController“. Falls ein Anwender allerdings innerhalb des Dialogs ein Primärschlüsselattribut entfernt oder einem Primärschlüsselattribut die Primärschlüsseleigenschaft entzieht, muss die Integrität der internen Repräsentation des Entity-Relationship-Diagramms von dieser Klasse gewährleistet werden. Dazu wird innerhalb dieser Klasse eine Referenz auf die Klasse „SketcherMainViewController“ gespeichert, über die dann der indirekte Zugriff auf die Geschäftslogikschicht realisiert wird, um den entsprechenden Geschäftsprozess ausführen zu können.

SketcherEntityTableModel Die Klasse „SketcherEntityTableModel“ erweitert die Klasse „DefaultTableModel“ des Java-Frameworks und wird im Zusammenhang mit der Steuerung der JTable verwendet, in der die einzelnen Attribute einer Entität dargestellt werden. Genauer gesagt wird innerhalb dieser Klasse festgelegt, welche Java-Datentypen in den einzelnen Spalten befinden und welche Zellen innerhalb der Tabelle editiert werden können.

Sketcher Relation Dialog Die Komponente „Sketcher Relation Dialog“ enthält, analog zu der vorherigen Komponente, Java-Elemente für das Bearbeiten einer Beziehung und die Steuerung des entsprechenden Dialogfensters (siehe Abbildung 36).

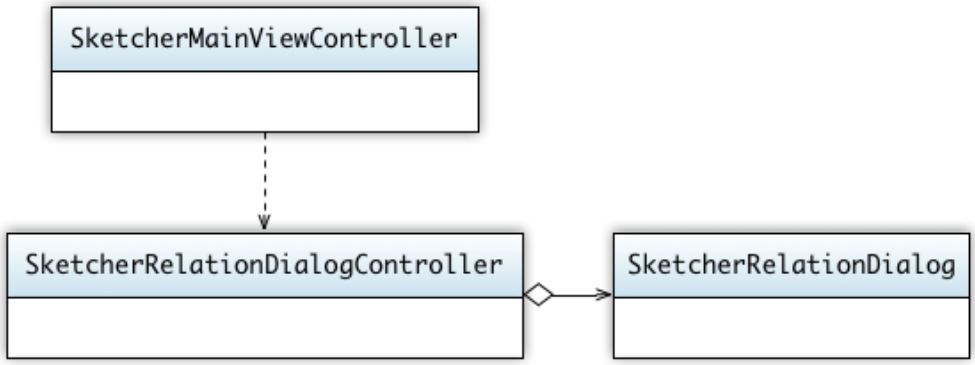


Abbildung 36: *Klassendiagramm zeigt die Elemente für die Benutzerinteraktion innerhalb des Dialogfensters zum Bearbeiten einer Beziehung.*

SketcherRelationDialog Diese Klasse stellt das Dialogfenster zum Bearbeiten einer Beziehung dar. Die Klasse beinhaltet alle in Abbildung 18 gezeigten GUI-Elemente. Diese Klasse beinhaltet auch keine fachlichen Aspekte, der Zugriff auf GUI-Elemente wird auch hier über Getter-Methoden gekapselt.

SketcherRelationDialogController Die Klasse „SketcherRelationDialogController“ steuert – wie alle bisher genannten Klassen mit dem Appendix „Controller“ – die GUI des Dialogfensters über die Klasse „SketcherRelationDialog“. Auch innerhalb dieser Klasse findet kein direkter Zugriff auf das

Ausführen von Geschäftsprozessen statt. Um eine Beziehung zu bearbeiten, wird lediglich eine Referenz der zu bearbeitenden Beziehung an diese Klasse übergeben, an der dann die Änderungen durchgeführt werden.

6.2.4 Sketcher Integration

Diese Schicht entspricht der Persistenz-/Datenhaltungsschicht des 3-Schichten-Modells und kapselt die Persistierung sowie das Abrufen von Geschäftsobjekten für die Schicht „Sketcher Business Logic“ (siehe Abbildung 37). Diese Schicht besteht momentan lediglich aus der Komponente „Sketcher Integrator“, da die Persistierung und das Abrufen von Geschäftsobjekten momentan lediglich über das JAXB-Framework angeboten wird. Falls in Zukunft für diese Aufgabe beispielsweise ein Datenbanksystem verwendet werden soll, wird diese Schicht mit Sicherheit um weitere (Sub-)Komponenten erweitert.

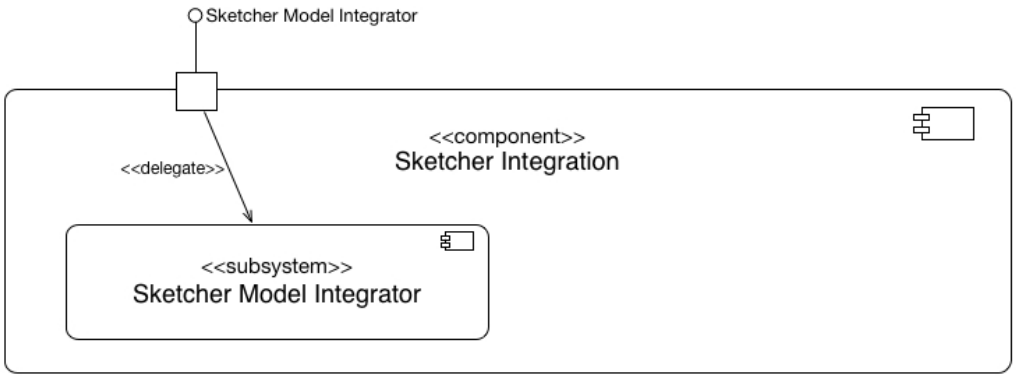


Abbildung 37: *“Sketcher Integration“-Schicht mit (Sub-)Komponente „Sketcher Model Integrator“.*

Sketcher Model Integrator Die Komponente „Sketcher Integrator“ beinhaltet das Java-Interface „SketcherModelIntegrator“ und die Klasse „SketcherModelJaxBIntegrator“. Sie stellt die zentrale Zugriffsstelle für die „Sketcher Business Logic“-Schicht dar (siehe Abbildung 38).

SketcherModelIntegrator Dieses Java-Interface stellt die Schnittstelle dieser Schicht dar, die nach Außen repräsentiert wird. Die Schicht „Sketcher Business Logic“ verwendet diese Schnittstelle, um Geschäftsobjekte zu

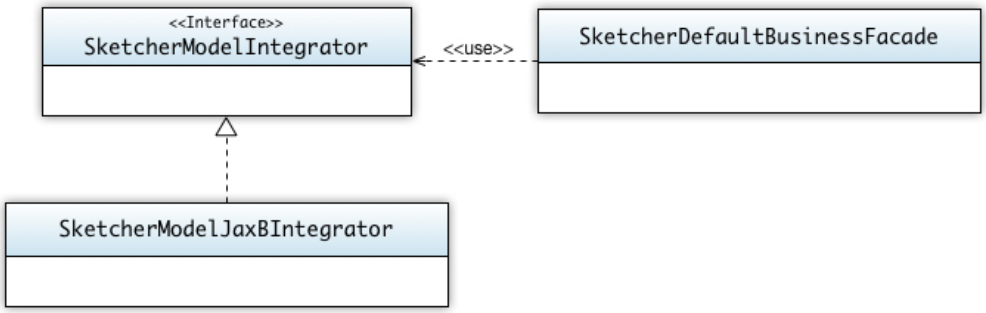


Abbildung 38: *Klassendiagramm zeigt die Elemente der "Sketcher Model Integrator"-Komponente.*

persistieren oder abzurufen. Auch hier ist für die "Sketcher Business Logic"-Schicht nicht relevant, welche konkrete Implementierung dieser Java-Schnittstelle letztlich verwendet wird.

SketcherModelJaxBIntegrator Die Klasse „SketcherModelJaxBIntegrator" ist die Standardimplementierung des im letzten Abschnitt eingeführten Java-Interface. Innerhalb der Klasse wird implementiert, wie genau die Geschäftsobjekte persistiert und wieder abgerufen werden können.

Hierfür wurde das JAXB-Framework[50] verwendet, das einfach ausgedrückt Java-Objekte in XML-Format umwandeln kann und vice versa. Dazu müssen die entsprechenden Geschäftsklassen[51], in einem ersten Schritt, mit speziellen Annotationen[52] des JAXB-Frameworks annotiert werden. Nach der Annotierung kann man durch das sogenannte „Marshalling", die Transformation von Objekten der Geschäftsklasse in das XML-Format veranlassen. Die Umkehroperation, also das Extrahieren von Geschäftsobjekten aus einer vorher gespeicherten XML-Datei, ist das sogenannte „Unmarshalling".

[50] siehe [Gla]
[51] Klassen der Komponente Sketcher Model, die den Geschäftsobjekten entsprechen.
[52] vgl. [Oraa]

7 Entwurfsmuster

Entwurfsmuster können einfach ausgedrückt als Katalog abstrakter Lösungen für verschiedene wiederkehrende Probleme bezüglich einer objektorientierten Softwarearchitektur verstanden werden. Sie sind im Laufe der Zeit seit der Einführung der objektorientierten Softwareentwicklung entstanden und bündeln die jahrelange Erfahrung vieler Entwickler.

Genauer gesagt zeigen Entwurfsmuster nicht nur Lösungen für bestimmte Probleme; deren Verwendung steigert auch die Qualität einer Architektur in Bezug auf Wiederverwendbarkeit, Anpassbarkeit und Erweiterbarkeit und damit generell die Pflegbarkeit. Allerdings soll nicht unerwähnt bleiben, dass die Verwendung von Entwurfsmustern zwar die Qualität einer Architektur in Bezug auf die eben genannten Eigenschaften erhöht, die Komplexität und der Abstraktionsgrad allerdings ebenfalls steigen. Diese Tatsache geht zu Lasten der Verständlichkeit. In diesem Zusammenhang sollen auch die sogenannten Anti-Patterns genannt sein, die ihrerseits bestimmte Muster für einen „schlechten Architekturstil“ zeigen (vgl. [ant]).

Ein weiterer wichtiger Aspekt ist die Tatsache, dass Entwurfsmuster auf einer relativ abstrakten Ebene beschrieben werden. Ein einzelnes Entwurfsmuster zeigt somit eine Lösung für eine ganze Reihe von Problemen einer bestimmten Kategorie unabhängig von der Programmiersprache, mit der die Software letztlich implementiert wird. Falls die Verwendung eines bestimmten Entwurfsmusters für die Lösung eines Entwurfsproblems in Frage kommt, stellt das Entwurfsmuster zunächst lediglich eine Art Vorgabe oder Gerüst für die tatsächliche Anwendung dar. Es muss an die vorherrschenden Gegebenheiten und Bedingungen des konkreten Problems angepasst werden. Hierbei sollte man pragmatisch statt dogmatisch vorgehen und ein Entwurfsmuster als Idee oder Hinweis für die Lösung eines Problems sehen, anstatt beispielsweise beharrlich zu versuchen, Teile eines Entwurfsmuster zu implementieren, die für das konkrete Problem nicht notwendig sind, nur weil diese in dem Entwurfsmuster vorgesehen sind. Die Modifikation eines Entwurfsmusters ist somit völlig legitim.

Da im letzten Abschnitt ein relativ detaillierter Einblick in die Architektur der zu entwickelnden Anwendung gegeben wurde, zeigt dieser Abschnitt, welche Entwurfsmuster für den Entwurf der Architektur verwendet wurden und was diese Tatsache im Einzelnen bedeutet. Die Entwurfsmuster werden dabei nicht in ihrer ursprünglichen Form, sondern in Bezug auf die bereits gezeigte Architektur beschrieben. Für die klassische Beschreibung von Ent-

wurfsmustern sei auf [GHJV 8] und [FF 6] verwiesen.

7.1 Model View Controller

Das „Model View Controller“-Entwurfsmuster ist sehr wichtig, da es einen Architekturentwurf in Bezug auf die Steuerung der grafischen Benutzeroberfläche und somit die Interaktion mit dem Anwender beschreibt. Das Muster sieht nach den bereits erwähnten Prinzipien „Separation of Concerns“und „Single Responsibilty“die Aufteilung in drei Elemente vor: Ein View, ein Modell und ein Controller (siehe Abbildung 39).

Das Model verwaltet gemäß Definition des Entwurfsmusters die Geschäftsobjekte, also die interne Repräsentation des Entity-Relationship-Diagramms. In diesem Fall wird das Model durch das Java-Interface „SketcherBusinessFacade“ realisiert.

Ein View beinhaltet lediglich technische Aspekte der GUI, ohne fachliche Aspekte zu implementieren, und wird hier von den Klassen „SketcherMainView“, „SketcherEntityDialog“ und „SketcherRelationDialog“ realisiert.

Der Controller hat die Aufgabe, über das View zwischen den Benutzerinteraktionen und dem Model zu vermitteln. Da für jedes View ein eigener Controller verwendet wird, werden die Controller im Rahmen dieser Arbeit von den Klassen „SketcherMainViewController“, „SketcherEntityDialogController“ und „SketcherRelationDialogController“ repräsentiert. Die letzten beiden genannten Controller können als „Sub-Controller“ des „SketcherMainViewController“ gesehen werden, da beide Controller von dem „SketcherMainViewController“ gestartet werden und keiner der beiden das Model direkt verwendet (Die Klasse „SketcherEntityDialogController“ stellt eine Ausnahme dar, greift aber auch nicht direkt sondern – wie bereits erwähnt – über die Klasse „SketcherMainViewController“ auf das Model zu).

Das Verwenden dieses Entwurfsmusters, bietet viele Vorteile. Ein Vorteil ist die Tatsache, dass die View-Klassen in einem anderen Kontext wiederverwendet werden können, da sie keine fachlichen Aspekte beinhalten. Des Weiteren kann die View-Klasse selbst verändert werden, um beispielsweise Elemente der grafischen Darstellung neu anzuordnen, ohne dass die Controller-Klasse angepasst werden muss. Die View-Klasse ist auch von Änderungen an der Implementierung von Geschäftsobjekten und Geschäftsprozessen nicht betroffen.

Ein weiterer Vorteil ist, dass das Model keine Kenntnis über die Controller hat, die es verwenden. Dadurch wirken sich Änderungen im Controller nicht

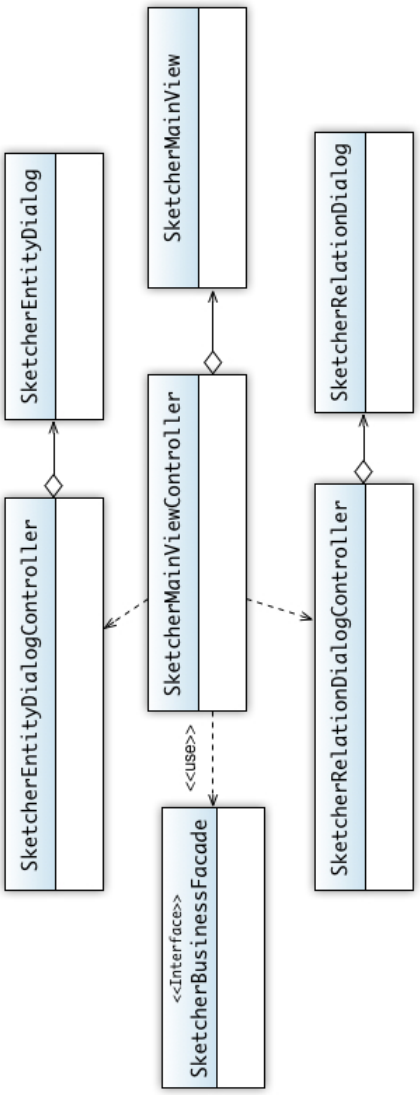


Abbildung 39: *Klassendiagramm zeigt die Elemente des „Model View Controller“-Entwurfsmusters innerhalb der Architektur für die zu entwickelnde Anwendung.*

auf das Model aus und es können beliebig viele Controller implementiert werden, ohne das Model anpassen zu müssen. Da das Model über die Java-Schnittstelle „SketcherBusinessFacade“ repräsentiert wird, kann die konkrete

Implementierung ausgetauscht oder verändert werden, ohne dass der Controller davon betroffen ist.

7.2 Facade

Das Facade-Entwurfsmuster beschreibt, wie die Verwendung eines Teilsystems[53] nach Außen zentral gekapselt werden kann. Dazu wird eine sogenannte Facade eingeführt, in der alle wichtigen Funktionen des Teilsystems, zentral implementiert werden. Die Verwendung des Teilsystems wird somit auf die Facade reduziert und die Implementierung vor Elementen verborgen, die das Teilsystem verwenden. Somit kann die Facade als Schnittstelle zu dem Teilsystem gesehen werden, auch wenn es sich dabei technisch **nicht** um ein Java-Interface handeln muss. Somit bietet eine Facade einen Vorteil, den alle Schnittstellen bieten. Nämlich die Tatsache, dass sich Änderungen in der Implementierung sich nicht auf Elemente auswirken, die das Teilsystem über die eingeführte Facade verwenden.

Im Rahmen dieser Arbeit wird die Facade durch das Java-Interface „SketcherBusinessFacade“ realisiert (siehe Abbildung 31). Da der letzte Abschnitt gezeigt hat, dass das Java-Interface „SketcherBusinessFacade“ das Model im Rahmen des „Model View Controller“-Entwurfsmusters darstellt, wurden hier zwei Entwurfsmuster kombiniert. Das Model verwaltet also nicht nur die Geschäftsobjekte, sondern stellt gleichzeitig auch eine schmale Schnittstelle für das Ausführen von Geschäftsprozessen dar; es kapselt somit die gesamte Geschäftslogik.

7.3 Strategy

Einfach ausgedrückt kann man durch Anwendung des Strategy-Entwurfsmusters die Verwendung eines Algorithmus von der konkreten Implementierung trennen. Dazu wird eine Schnittstelle in Form einer abstrakten Java-Klasse oder eines Java-Interface eingeführt, in der lediglich die Methode (Strategy-Methode) für das Ausführen des Algorithmus implementiert wird. Für die Realisierung der konkreten Algorithmen muss die abstrakte Java-Klasse erweitert oder das Java-Interface implementiert und die Strategy-Methode mit dem konkreten Algorithmus gefüllt werden. Für Elemente, die

[53] Ein Verbund aus Klassen etc., die ein zusammenhängendes Teilsystem formen; kann auch als Komponente gesehen werden.

den Algorithmus verwenden, ist es durch das Einführen der Schnittstelle nicht mehr relevant, welche konkrete Implementierung des Algorithmus verwendet wird. Die konkreten Algorithmen können somit beliebig ausgetauscht und erweitert werden.

Das Strategy-Entwurfsmuster wird innerhalb der Anwendung im Wesentlichen zweimal verwendet. Das Strategy-Entwurfsmuster wurde einerseits für die Persistierung und das Abrufen von Geschäftsobjekten („Sketcher Model Integrator“, siehe Abschnitt 6.2.4) verwendet. Andererseits wird das Generieren von SQL-Code ebenfalls durch das Strategy-Entwurfsmuster realisiert.

8 Erweiterung und Anpassung

In den letzten Abschnitten wurde teilweise erwähnt, was die Einteilung der Softwarearchitektur in Schichten und Komponenten sowie die Verwendung der beschriebenen Entwurfsmuster für die Erweiterbarkeit und Anpassung der Architektur bedeuten. Dieser Abschnitt zeigt mögliche Lösungen, um bestimmte Aspekte der bestehenden Architektur zu erweitern bzw. anzupassen.

Vorab soll dabei erwähnt sein, dass neben den im Folgenden gezeigten Ansätzen für die Erweiterung/Anpassung in Bezug auf die visuelle Darstellung mithilfe des JGraphX-Frameworks auch immer die Möglichkeit besteht, das Framework genaustens zu analysieren, um tiefergehende Änderungen vornehmen zu können[54]. Da eine tiefergehende Auseinandersetzung mit dem Framework äußerst zeitaufwendig ist und im Rahmen der Entwicklung nur ein begrenzter Zeitrahmen zu Verfügung stand, wurde bei der Entwicklung versucht, so wenig Aufwand wie möglich in die Verwendung des JGraphX-Frameworks zu investieren. So wurde vorrangig versucht, Funktionen des Framworks zu **benutzen**, anstatt sie zu **verändern**. Ein Beispiel dafür ist die Darstellung einer Entität, die durch eine pragmatische Lösung mithilfe der vorhandenen Elemente des JGraphX-Frameworks, statt durch Veränderung der Elemente, bewerkstelligt wurde.

[54]Mit tiefergehenden Änderungen ist hier gemeint, dass beispielsweise wichtige Klassen des Frameworks erweitert werden, um Basisfunktionalitäten zu überschreiben.

8.1 Krähenfuß-Notation

Das Anpassen oder Erweitern der Darstellung von Beziehungen gemäß der Krähenfuß-/IE-Notation entsprechend Anforderung F70 gestaltet sich als relativ schwierig, weswegen auch für die erste Version der zu entwickelnden Anwendung die in Anforderung F20 gezeigte Notation gewählt wurde.

Ein Ansatz, die Darstellung von Beziehungen anzupassen ist, eine neue ,bereits in Abschnitt 4.3.2 erwähnte Dia-Shape-Datei für die jeweilige Darstellung der verschiedenen Beziehungstypen gemäß der Krähenfuß-/IE-Notation zu erzeugen. Allerdings wurde bereits in Abschnitt 6.2.3 erläutert, dass dieser Ansatz nicht umgesetzt werden konnte, da mehrere Elemente nicht zu einer neuen Form zusammengefasst werden konnten.

Denselben Ansatz wie bei der Darstellung einer Entität zu wählen und verschiedene Linien zu einer speziellen Beziehung der Krähenfuß-/IE-Notation zu einer Gruppe zusammenzufassen, ist grundsätzlich möglich. Der Nachteil diese Methode ist, dass immer auch der Elternknoten als Rahmen um die Kindknoten dargestellt wird. Falls eine Möglichkeit existiert, den Elternknoten unsichtbar zu machen, wäre dies ein Ansatz, um die Beziehungen entsprechend der Krähenfuß-/IE-Notation zu visualisieren.

8.2 Objektrelationalität

In Bezug auf die Darstellung der objektrationalen Beziehungen „IS-A“ und „ist Teil von“ siehe Abbildung 4 und 5 (gemäß Anforderung F80) gilt dasselbe, wie für die Darstellung von Beziehungen entsprechend der Krähenfuß-/IE-Notation: Die Veränderung der Darstellung von Beziehungen mithilfe von Dia-Shape-Formen konnte nicht umgesetzt werden und der Ansatz der Gruppierung funktioniert nur, falls der Elternknoten unsichtbar gemacht werden kann.

Das Erweitern der Anwendung in Bezug auf die Möglichkeit, eigene SQL-Typen zu definieren, ist grundsätzlich denkbar. Dazu könnte zunächst ein Button in dem Dialogfenster für das Bearbeiten einer Entität (siehe Abbildung 17) und ein Dialogfenster für das Definieren eines eigenen SQL-Typs hinzugefügt werden. Der Mausklick auf den Button startet den neuen Dialog, in dem der Anwender den neuen SQL-Typ definieren kann. Auf der fachlichen Seite muss ein Mechanismus implementiert werden, um neue SQL-Typen aufnehmen zu können (dazu gehört die Abbildung der Geschäftsobjekte auf Java-Klassen sowie die Implementierung der entsprechenden Geschäftspro-

zesse).

Zusätzlich dazu muss die Java-Enumeration, auf der die unterstützten SQL-Datentypen abgebildet sind, ersetzt oder zumindest angepasst werden. Grund dafür ist, dass die Enumeration verwendet wird, um die unterstützten SQL-Datentypen innerhalb des Dialogs für das Bearbeiten einer Entität, als Auswahl für den Datentyp eines Attributs anbieten zu können. Von dem Anwender neu definierte SQL-Datentypen müssen nach ihrer Definierung ebenfalls als mögliche Auswahl zur Verfügung stehen. Da die Java-Enumeration, auf der die SQL-Datentypen abgebildet sind, in ihrer derzeitigen Form nicht für das dynamische Hinzufügen von Enumerationen zu gebrauchen ist, muss sie entweder diesbezüglich erweitert oder ein separates Element für diese Aufgabe eingeführt werden.

Des Weiteren müssen die neuen SQL-Typen bei dem Speichern eines Entity-Relationship-Diagramms ebenfalls gespeichert und beim Laden ebenfalls wieder geladen werden. Dazu müsste die veränderte Java-Enumeration, oder das neu eingeführt Element, lediglich mit den bereits erwähnten Annotationen des JAXB-Frameworks annotiert werden. Bezüglich der Generierung von SQL-Code, müsste die Definition des neu eingeführten SQL-Typs vor der Definition von Tabellen eingeführt werden.

8.3 Zusätzliche SQL-Dialekte

Die Anwendung soll hauptsächlich das Generieren von Oracle- und MySQL-Code unterstützen. Es ist allerdings durchaus möglich, dass in Zukunft noch weitere SQL-Dialekte unterstützt werden sollen. Wie bereits erwähnt wird das Generieren von SQL-Code einer konkreten Datenbanktechnologie über die Verwendung des Strategy-Entwurfsmusters realisiert (siehe Abbildung 40).

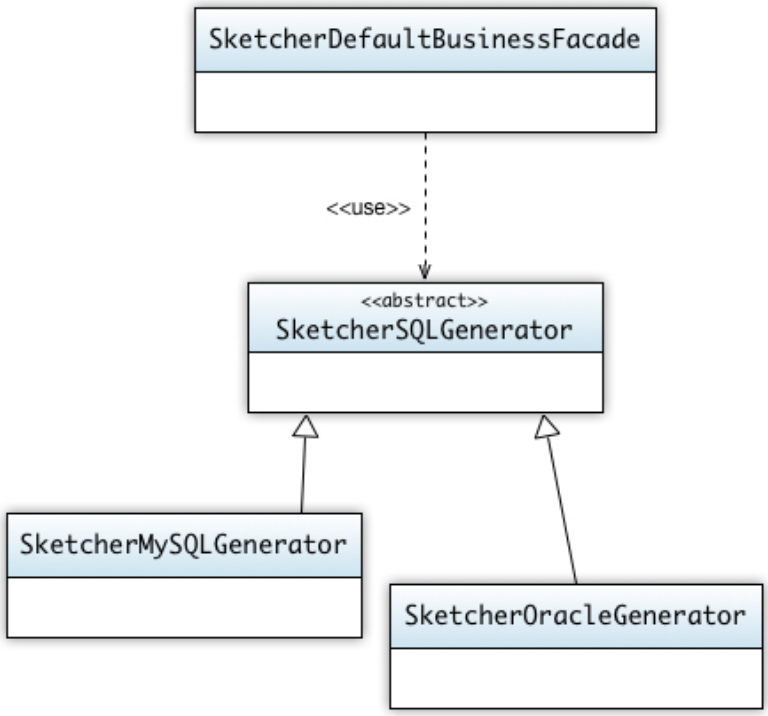


Abbildung 40: *Klassendiagramm zeigt die Elemente des Strategy-Entwurfsmuster für das Generieren von konkretem SQL-Code.*

Wie bereits erwähnt kümmert sich die abstrakte Klasse „SketcherSQL-Generator“ um den Großteil der Generierung von SQL-Code. Lediglich für die konkrete Zuordnung der abstrakten SQL-Datentypen zu konkreten datenbankspezifischen SQL-Code benutzt die Klasse eine abstrakte Methode (matchDatatype), die von Unterklassen implementiert werden muss. Falls ein neuer Dialekt eingeführt werden soll, muss also zunächst eine Klasse eingeführt werden, die ihrerseits die abstrakte Klasse „SketcherSQLGenerator“ erweitert und die Strategy-Methode implementiert, um die Datentypen des neuen Dialekts zuzuordnen. Als nächstes wird ein entsprechender Eintrag in der Spring-Konfiguration (wird in Abschnitt 9.2 noch genauer erklärt) und eine entsprechende Properties-Datei[55] benötigt. Die Properties-Datei enthält

[55]In sogenannten Properties Dateien können Einstellungen oder Konfigurationen eingetragen werden. Ein Eintrag besteht aus einer Bezeichnung und einem Wert.

konstante Standardwerte für die Parameter der datenbankspezifischen Datentypen (Beispielsweise benötigt der MySQL-Datentyp VARCHAR einen Parameter, der die Länge des zu speichernden Textes beschreibt) und trägt dieselbe Bezeichnung wie die neue eingeführte Klasse, mit dem Appendix „.default“. Die Standardwerte müssen dann innerhalb der Implementierung der Strategy-Methode in der neu eingeführten Klasse entsprechend berücksichtigt werden.

Es soll hier noch gesagt sein, dass einzelne Methoden der abstrakten Klasse „SketcherSQLGenerator“ in Unterklassen überschrieben werden können, um einzelne Aspekte, beispielsweise die Generierung des Fremdschlüssels, an eine konkrete Datenbanktechnologie anzupassen.

8.4 Persistierung und Abrufen ändern

Da das Persistieren und Abrufen eines gezeichneten Entity-Relationship-Diagramm ebenfalls durch das Strategy-Entwurfsmuster realisiert wurde, gestaltet sich das Einführen einer neuen Methode zum Persistieren und Abrufen relativ einfach. Dazu muss lediglich eine neue Klasse eingeführt werden, die das Java-Interface „SketcherModelIntegrator“ und die entsprechenden Methoden implementiert. Damit die Anwendung weiß, welche konkrete Implementierung verwendet werden soll, wird hier ebenfalls eine Anpassung der Spring-Konfiguration benötigt.

9 Technologien

Im letzten Abschnitt wurde eine relativ dataillierte Übersicht der Architektur sowie Erweiterungs- und Anpassungsmöglichkeiten gegeben. In diesem Abschnitt werden die Technologien, die im Rahmen der Entwicklung eingesetzt werden, und deren Verwendung beschrieben, da sie teilweise im Zusammenhang zur Erweiterbarkeit bzw. Anpassung stehen.

9.1 Maven

Maven[56] ist ein Tool oder Framework, das in erster Linier verschiedene Abhängigkeiten eines Softwareprojekts zu verschiedenen anderen Frameworks

[56] vgl. [Apab]

oder Softwareprojekten verwaltet. Darüber hinaus bietet Maven die Möglichkeit, verschiedene Aspekte eines Softwareprojekts zu automatisieren. Als Beispiel soll hier das Kompilieren des Codes, das Erzeugen von Dokumentationen oder einer ausführbaren JAR-Datei und das Ausführen von Tests genannt sein. Im Rahmen dieser Arbeit wird das Maven für die folgenden Aspekte verwendet.

Abhängigkeit zu JGraphX Abhängigkeiten zu anderen Projekten oder Frameworks werden in der Maven-Konfigurationsdatei (pom.xml) eines Softwareprojekts definiert. So gut wie jedes größere oder oft verwendete Framework kann über das sogenannte Maven Central Repository[57] in ein eigenes Projekt integriert werden, indem ein entsprechender Eintrag in die Maven-Konfigurationsdatei vorgenommen wird. Leider wurde das JGraphX-Framework nicht in das Maven Central Repository aufgenommen und laut Angaben der Entwickler wird das wohl auch so bleiben.

Wenn man das JGraphX-Framework dennoch innerhalb eines eigenen Projekts verwenden möchte, muss man sich den Quellcode herunterladen und in ein Projekt innerhalb der bevorzugten Entwicklungsumgebung laden. Um das JGraphX-Framework in einem eigenen Projekt benutzen zu können, muss innerhalb der Entwicklungsumgebung eine Abhängigkeit zu dem JGraphX-Framework definiert werden. Der Nachteil dieser Methode ist, dass die Abhängigkeit nicht in den Quelldateien des Projekts, sondern in dem Projekt selbst durch die jeweilige Entwicklungsumgebung gespeichert wird. Werden diese Quelldateien des eigenen Projekts beispielsweise in eine andere Entwicklungsumgebung importiert, geht die Information bezüglich der Abhängigkeit verloren.

Daher wurde im Rahmen dieser Arbeit entschieden, das JGraphX Projekt zu einem Maven-Projekt zu erweitern. Dazu wurde das Projekt im Wesentlichen um eine Maven-Konfigurationsdatei erweitert (siehe Listing 1). Somit kann durch den entsprechenden Maven-Befehl (mvn install, in dem Hauptordner des Softwareprojekts) eine JAR-Datei erzeugt werden, die alle Quell-Dateien des JGraphX-Frameworks enthält und in anderen Projekten über den entsprechenden Eintrag deren Maven-Konfigurationsdatei, als Abhängigkeit definiert werden. Durch den Befehl wird die JAR-Datei nicht nur erzeugt,

[57] Das Maven Central Repository ist einfach ausgedrückt eine zentrale Sammelstelle für die verschiedensten Frameworks.

sie wird ebenfalls in das lokale Maven Repository[58] abgelegt. Ohne genauer auf einzelne Aspekte dieses Vorgangs eingehen zu wollen, hat diese Tatsache unter anderem den Vorteil, dass die Quell-Dateien des JGraphX-Frameworks nicht mehr lokal verfügbar sein müssen um Abhängigkeiten zu erzeugen. Die durch Maven generierte JAR-Datei muss sich lediglich in dem lokalen Maven Repository befinden. Falls sich die Quell-Dateien des JGraphX-Frameworks sich im Laufe der Zeit[59] ändern, muss die JAR-Datei erneut generiert werden (mvn install), damit die abhängenden Projekte – einfach gesagt – auch auf die geänderte Version zugreifen können.

Listing 1: *Maven-Konfigurationsdatei für das JGraphXProjekt.*

```
<project ... >
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.sketcher</groupId>
    <artifactId>JGraphXMaven</artifactId>
    <version>SNAPSHOT</version>
    <build>
        <sourceDirectory>src/</sourceDirectory>
        <resources>
            <resource>
                <directory>src/com/mxgraph/swing/images</
                    directory>
                <includes>
                    <include>**/*.gif</include>
                </includes>
                <targetPath>com/mxgraph/swing/images</targetPath>
            </resource>
        </resources>
    </build>
</project>
```

Die Maven-Konfigurationsdatei des JGraphX-Maven-Projekts (siehe Listing 1) enthält folgende wichtigen Elemente: Die „groupId“ ist eine Id[60], die für verschiedene Maven-Projekte einer Softwareprodukts immer gleich sein muss. Die „artifactId“ ist der Name des Projekts und die „version“ ist das Appendix, was der generierten JAR-Datei (bei Ausführen des Befehls „mvn install“) hinzugefügt wird. Innerhalb des „build Blocks“ werden hier einige Bilder, die sich innerhalb der Quell Dateien des JGraphX-Frameworks befin-

[58] Werden Abhängigkeiten zu Frameworks in einer Maven-Konfigurationsdatei definiert, so sucht Maven zunächst im lokalen und danach im Central Maven Repository danach.

[59] Beispielsweise im Zuge der Weiterentwicklung.

[60] Eindeutige Nummer.

den explizit in die zu erzeugende JAR-Datei „kopiert“. Grund dafür ist die Tatsache, dass Maven standardmäßig nur Java-Klassen und Dateien aus dem Verzeichnis „src/main/ressources“ in die zu erzeugende JAR-Datei generiert, allerdings benötigen einige der Basiselemente des JGraphX-Frameworks die eben erwähnten Bilder, um aufgerufen werden zu können.

Ausführbare JAR-Datei In dem Projekt für die zu entwickelnde Anwendung muss die Abhängigkeit zu dem Maven-Projekt des JGraphX-Framework – wie bereits erwähnt – in die entsprechende Maven-Konfigurationsdatei aufgenommen werden (siehe Listing 2). Neben der Definition von Abhängigkeiten wird Maven für dieses Projekt auch verwendet um eine ausführbare JAR-Datei der Anwendung samt Abhängigkeiten zu erzeugen (mvn install package). Bei der Erzeugung der ausführbaren JAR-Datei werden neben dem Quellcode und den Abhängigkeiten auch verschiedene Konfigurationsdateien (siehe Abschnitt 10.1) einbezogen.

Listing 2: *Maven-Konfigurationsdatei für das Projekt der zu entwickelnden Anwendung.*

```
<project ... >
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.sketcher</groupId>
    <artifactId>SketchER</artifactId>
    <version>ALPHA</version>
    <dependencies>
        ...
        <dependency>
            <groupId>de.sketcher</groupId>
            <artifactId>JGraphXMaven</artifactId>
            <version>SNAPSHOT</version>
        </dependency>
    </dependencies>
    <build>
        <resources>
        ...
        </resources>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <executions>
                    <execution>
                        <id>copy-dependencies</id>
```

```
25                      <phase>prepare-package</phase>
26                      <goals>
27                          <goal>copy-dependencies</goal>
28                      </goals>
29                      <configuration>
30                          <outputDirectory>SketchER/lib</
                                outputDirectory>
31                          ...
32                      </configuration>
33                  </execution>
34              </executions>
35          </plugin>
36          <plugin>
37              <groupId>org.apache.maven.plugins</groupId>
38              <artifactId>maven-jar-plugin</artifactId>
39              <version>2.3.1</version>
40              <configuration>
41                  <outputDirectory>SketchER</outputDirectory>
42                  <archive>
43                      <manifest>
44                          ...
45                          <mainClass>de.sketcher.Main.
                                SketcherMainEntry</mainClass>
46                      </manifest>
47                  </archive>
48              </configuration>
49          </plugin>
50      </plugins>
51  </build>
52 </project>
```

9.2 Spring

Die Verwendung des Spring[61]-Frameworks wurde im Rahmen dieser Arbeit schon in Bezug auf die Erweiterung und Anpassung der Anwendung (siehe Abschnitt 8) durch das Erwähnen der Spring-Konfigurationsdatei angedeutet. Das Spring-Framework ist ein sehr mächtiges Java-Framework, das hauptsächlich für die Entwicklung von "Java-Enterprise-Edition"-Anwendungen verwendet werden kann. Auf die vielen Aspekte des Spring Frameworks wird im Rahmen dieser Arbeit nicht eingegangen, für eine allgemeine Einführung der verschiedenen Aspekte des Spring-Frameworks soll auf

[61] vgl. [Piv]

[Wol20] verwiesen sein. Im Rahmen dieser Arbeit wird das Spring-Framework lediglich für die sogenannte Dependency Injection[62] verwendet. Einfach ausgedrückt kann man mithilfe der einer Dependency Injection relativ komfortabel konfigurieren, welche konkrete Klasse einer Schnittstelle (Java-Interface oder abstrakte Java-Klasse) zur Laufzeit instantiiert werden soll. Es dürfte sich von selbst verstehen, dass die zu instanziierende Java-Klasse das Java-Interface implementieren oder die abstrakte Java-Klasse erweitern muss. Die Konfiguration wird dabei in einer Spring-Konfigurationsdatei definiert. Diese Tatsache bietet den großen Vorteil, dass die Entscheidung, welche konkrete Klasse verwendet werden soll, nicht mehr explizit innerhalb des Java-Codes implementiert werden muss, sondern separat in der Spring-Konfigurationsdatei gekapselt wird (siehe Listing 3).

Listing 3: *Spring Konfigurationsdatei für das Projekt der zu entwickelnden Anwendung.*

```
1 <beans ... >
2     <context:annotation-config />
3
4     <bean id="businessFacade"
5      class="de.sketcher.facade.SketcherDefaultBusinessFacade">
6     </bean>
7     <bean id="integrator"
8      class="de.sketcher.integration.SketcherModelJaxBIntegrator
           ">
9     </bean>
10
11    <bean id="mysql"
12     class="de.sketcher.generator.SketcherMySQLGenerator">
13    </bean>
14    <bean id="oracle"
15     class="de.sketcher.generator.SketcherOracleGenerator">
16    </bean>
17 </beans>
```

Innerhalb des Java-Codes muss lediglich der sogenannte „Application-Context[63]“ initialisiert werden, der seinerseits die in Listing 3 gezeigte Spring-Konfigurationsdatei lädt. Über den „Application-Context“ kann dann die konkrete Klasse einer Schnittstelle durch das Spring-Framework instantiiert

[62] siehe [Fow]

[63] Der Application-Context sorgt hauptsächlich dafür Spring-Beans zu instantieren. Alle Java-Klassen die von Spring instantiiert werden, können als Spring-Beans bezeichnet werden.

werden. Wird beispielsweise nach der Spring-Bean „businessFacade“ gesucht, so instantiiert der „Application-Context“ die Klasse „SketcherDefaultBusinessFacade“. Falls die konkrete Klasse ausgetauscht werden sollte, so muss die Klasse nicht nur die entsprechende Schnittstelle implementieren sondern es muss zusätzlich der voll qualifizierte Name der neuen Klasse in der entsprechenden Stelle der Spring-Konfigurationsdatei anstelle der alten Klasse eingetragen werden.

9.3 Log4J

Bei dem Log4J-Framework (vgl. [Apaa]) handelt es sich um ein Java-Framework, das innerhalb von Java Anwendungen für das sogenannte Logging[64] verwendet werden kann. Das Konzept des Loggings hat im Rahmen dieser Arbeit einen besonderen Stellenwert. Um viele der nichtfunktionalen Anforderungen (siehe Abschnitt 3.2) zu gewährleisten, können verschiedene Softwaretests (Unit-Tests[65], Integrationstest[66] etc.) verwendet werden. Leider nimmt das Realisieren und Pflegen der entsprechenden Tests sehr viel Zeit in Anspruch, weshalb die Entscheidung getroffen wurde, die Softwaretests zu vernachlässigen und stattdessen ausführliche Logging-Ausgaben zu generieren.

Listing 4: *Log4J-Konfigurationsdatei erzeugt Logging-Ausgaben des Log-Levels DEBUG auf der Konsole.*

```
1 # Root logger option
2 log4j.rootLogger=DEBUG, stdout
3
4 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
5 log4j.appender.stdout.Target=System.out
6 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
7 log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd
      HH:mm:ss} %-5p %c{1}:%L - %m%n
8
9 log4j.logger.org.springframework = INFO
```

[64] Das Logging ist eine Technik um den Programmablauf zu protokollieren.

[65] Mit Unit-Tests testet man eine alleinstehende Einheit, beispielsweise eine Java-Klasse.

[66] Mit Integrationstests testet man das Zusammenspiel verschiedener alleinstehender Einheiten.

Die Logging-Ausgaben ersetzen zwar nicht die Softwaretests, sind aber ein pragmatisches, weniger zeitaufwendiges Provisorium, da sie das Debugging[67] stark vereinfachen.

Der Vorteil der Verwendung eines Logging-Frameworks wie Log4J ist die komfortable Konfiguration der Loggings über eine Log4J-Konfigurationsdatei (siehe Listing 4). Dabei kann die Ausgabe (auf Konsole oder in Datei) wie auch die Formatierung der verschiedenen Loggings festgelegt werden. Des Weiteren lassen sich mit Log4J Loggings verschiedener sogenannter Log-Level erzeugen, was den Vorteil bietet, dass je nach Bedarf die Ausgabe der Loggings einem gewünschten Log-Level entsprechend gefiltert werden kann.

10 SketchER

Die Tatsache, dass die im Rahmen dieser Arbeit entwickelte Anwendung den Namen „SketchER“ trägt, dürfte bereits durch die entsprechenden Appendices verschiedener Elemente der Architektur erahnt werden. Die vorherigen Abschnitte haben die Architektur der im Rahmen dieser Arbeit entwickelten Anwendung in der sogenannten ALPHA-Version gezeigt. Es könnte durchaus sein, dass die Anwendung mittlerweile bereits weiterentwickelt wurde und sich daher in einem aktuelleren Zustand befindet.

Der folgende Abschnitt zeigt die Struktur des Projekts aus der Sicht der Entwicklungsumgebung Eclipse[68] und einige Screenshots der Anwendung.

10.1 Projektstruktur

Abbildung 41 zeigt die Projektstruktur der Anwendung innerhalb der Entwicklungsumgebung Eclipse. Alle (abstrakten) Java-Klassen und -Schnittstellen befinden sich in den Verzeichnissen „src/main/java“ und „src/test/java“, wobei sich in dem letzteren Verzeichnis Java-Klassen befinden, die sogenannte Unit-Tests mit dem JUnit Framework[69] realisieren. In die Verzeichnisse „src/main/resources“ und „src/test/resources“ können Dateien abgelegt werden, die in den sogenannten Classpath[70] der Anwendung aufgenommen

[67] Der Prozess zum Finden und Beheben von Fehlern im Rahmen der Softwareentwicklung wird Debugging genannt.

[68] vgl. [Ecl]

[69] siehe [jun]

[70] Der Classpath, zu deutsch Klassenpfad, ist ein Pfad in dem der Java Compiler nach Java-Klassen sucht.

werden sollen. Beispiele für Dateien, die im Classpath verfügbar sein müssen, sind sämtliche Dateien zur Konfiguration der Anwendung, also Dateien mit dem Appendix „.properties“, „.default“ oder „.xml“.

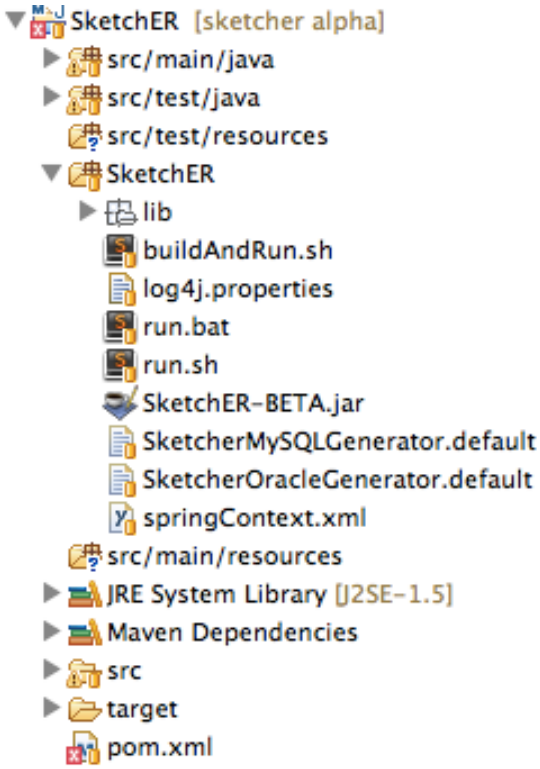


Abbildung 41: *Projektstruktur.*

Im Rahmen dieser Anwendung wurde das Verzeichnis „SketchER“ innerhalb des Projekts angelegt, in dem alle bereits erläuterten Konfigurationsdateien (außer die Maven-Konfigurationsdatei, die sich im Hauptverzeichnis des Projekts befindet) sowie eine ausführbare JAR-Datei und entsprechende Start-Skripte der Anwendung enthalten. Um die Anwendung zu starten, muss die Datei run.sh auf Unix-Betriebssystemen, die Datei run.bat auf Windows-Betriebssystemen innerhalb dieses Verzeichnisses aufgerufen werden. Das Skript buildAndRun.sh ist ein besonderes Skript, das zunächst mit Maven eine ausführbare JAR-Datei in das Verzeichnis „SketchER“ generiert. Dabei werden die Einstellungen in den Konfigurationsdateien übernommen. Anschließend wird die Anwendung gestartet. Man muss dieses Skript ver-

wenden, falls man Änderungen in den Konfigurationsdateien vorgenommen hat, wobei Maven auf dem Rechner installiert sein muss. Bislang wurde leider keine Möglichkeit gefunden, Änderungen in den Konfigurationsdateien, ohne das anschließend benötigte Ausführen der Maven-Befehle zu übernehmen.

10.2 Screenshots

In diesem Abschnitt werden einige Screenshots der Anwendung gezeigt. So zeigt Abbildung 42 die Darstellung einer Entität und Abbildung 43 den Dialog zum Bearbeiten einer Entität. Abbildung 44 zeigt die Darstellung einer 1-n-Beziehung zwischen zwei Entitäten und Abbildung 45 zeigt den Dialog zum Bearbeiten einer Beziehung. Abschließend wird in Abbildung 46 die Darstellung einer n-m-Beziehung gezeigt.

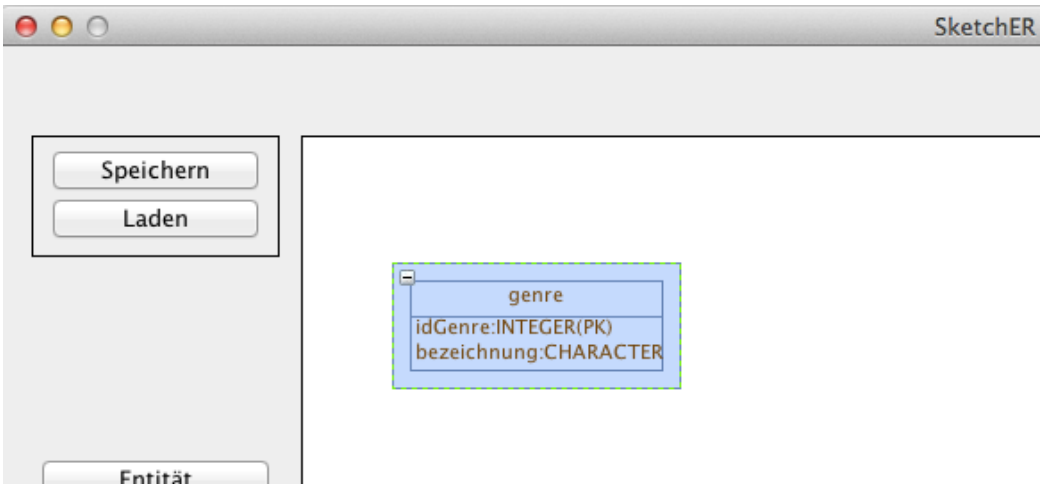


Abbildung 42: *Darstellung einer Entität.*

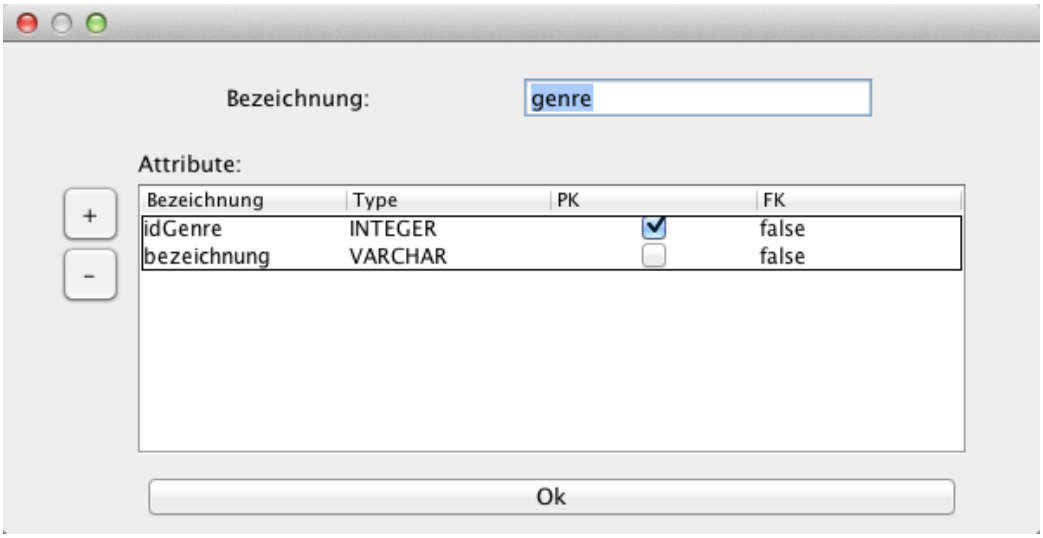


Abbildung 43: *Dialog für das Bearbeiten einer Entität.*

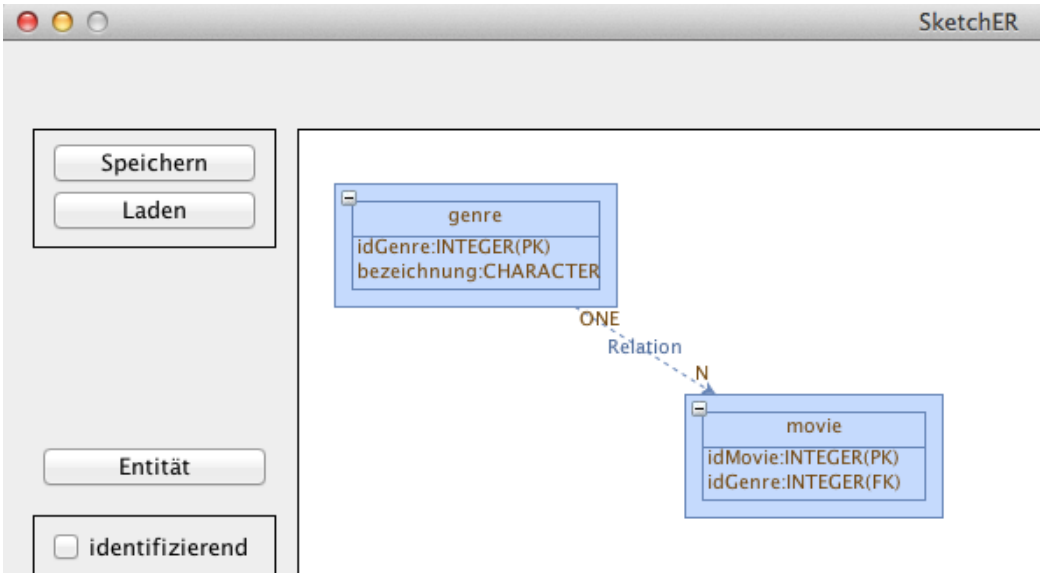


Abbildung 44: *Darstellung einer nicht identifizierenden 1-n-Beziehung zwischen zwei Entitäten.*

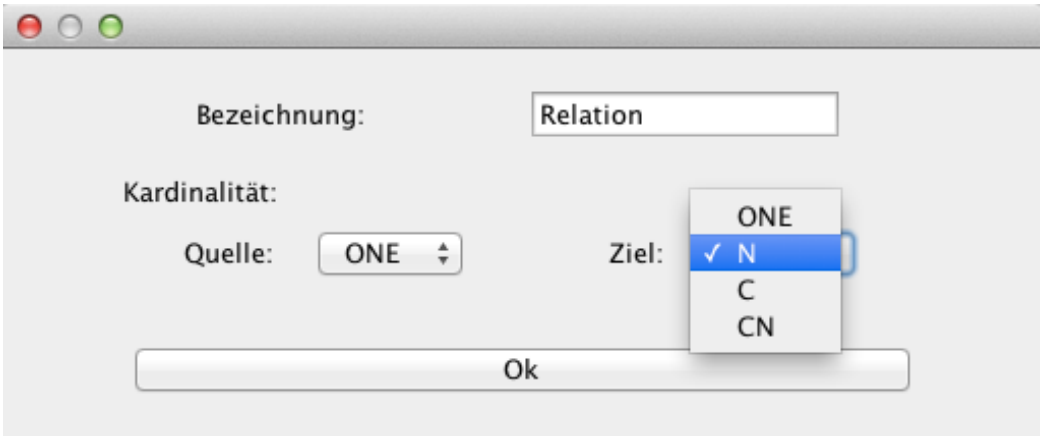


Abbildung 45: *Dialog zum Bearbeiten einer Beziehung.*

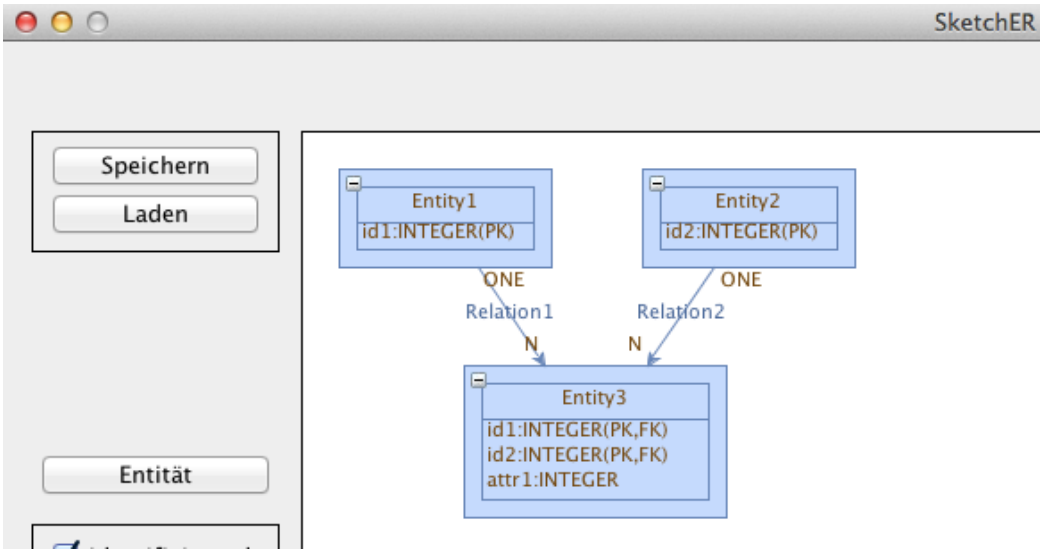


Abbildung 46: *Darstellung einer n-m-Beziehung mit Zwischentabelle.*

11 Schlussbetrachtung

Im Rahmen dieser Arbeit wurde der Prototyp (ALPHA-Version) einer Anwendung zum Zeichnen von Entity-Relationship-Diagrammen implementiert und vollständig dokumentiert. Innerhalb der ALPHA-Version können Anwender Entitäten zeichnen, löschen sowie deren Attribute bearbeiten, wo-

bei diese der Anforderung F10 entsprechend dargestellt werden. Weiterhin können binäre (identifizierende) Beziehungen zwischen Entitäten erzeugt und gelöscht (F20) und Entitäten beliebig innerhalb der JPanels bewegt werden (F30). Bezüglich F30 soll gesagt sein, dass Anwender Entitäten nur dann bewegen können, wenn der in Abschnitt 6.2.3 genannte Elternknoten der Entität bewegt wird. Das Bewegen einer Entität über den Kopfbereich oder den Bereich, in dem die Attribute dargestellt werden, funktioniert leider nicht. Da dieses Verhalten in der ALPHA-Version nicht verbessert werden konnte, wird empfohlen, Entitäten nur im zusammengeklappten Zustand zu bewegen.

Das Speichern und Laden eines Entity-Relationship-Diagramms entsprechend F40 wurde zwar umgesetzt, allerdings ist diese Funktionalität innerhalb der ALPHA-Version nicht ganz ausgereift. Wenn man ein Entity-Relationship-Diagramm aus einer vorher gespeicherten Datei lädt, so zeigt die Anwendung teilweise invalides, nicht nachvollziehbares Verhalten. Das Exportieren eines Bilds (F60) und das Generieren von SQL-Code (F60) aus einem gezeichneten Entity-Relationship-Diagramm wurde in der ALPHA-Version ebenfalls umgesetzt, wobei die wichtigsten Fälle unter MySQL und Oracle getestet wurden.

Bezüglich der nichtfunktionalen Anforderungen kann gesagt werden, dass N10 in Bezug auf alle funktionalen Anforderungen außer F40 zutrifft, da alle funktionalen Anforderungen wie spezifiziert umgesetzt werden konnten. Bezüglich N20 wurden an vielen Stellen Warnungen und Optionsdialoge eingebaut, beispielsweise um zu bestätigen, dass ein Element wirklich gelöscht werden soll, oder die Warnung, dass zwei Attribute mit derselben Bezeichnung innerhalb einer Entität nicht angelegt werden können, ebenso wenig Attribute ohne Bezeichnung und Datentyp. Zu der Zuverlässigkeit in Bezug auf N20 kann keine genaue Angabe gemacht werden, da die Anwendung aufgrund des begrenzten Zeitraums für diese Arbeit nicht intensiv getestet werden konnte. Ob die Anwendung nach längerer Benutzung immer noch zuverlässig funktioniert und valides Verhalten zeigt, wird sich erst zeigen, wenn die Anwendung produktiv und intensiv genutzt wird.

Da die grafische Oberfläche schlicht gehalten wurde und nur Elemente enthält, die wirklich notwendig sind, kann gesagt werden, dass die Anwendung ohne weiteren Aufwand durch den Anwender benutzt werden kann, auch wenn hier nicht explizit Methoden aus dem Bereich Usability-Engineering verwendet wurden.

Im Rahmen der Entwicklung der ALPHA-Version lag der Schwerpunkt auf der Umsetzung der primären funktionalen Anforderungen, neben der Um-

setzung der nichtfunktionalen Anforderungen. Allerdings soll die Anwendung zukünftig um die sekundären funktionalen Anforderungen erweitert werden können. In Bezug auf N40 wurden daher zunächst Möglichkeiten der Erweiterung/Anpassung in Bezug auf die sekundären Anforderungen in Abschnitt 8 gezeigt. Weiterhin wurde bei der Implementierung darauf geachtet, die Wartbarkeit der Anwendung zu gewährleisten, unter anderem durch Dokumentation des Codes, Verwendung von Entwurfsmustern und allgemein durch Anwenden von bewährten Design-Prinzipien der Softwareentwicklung.

Abschließend kann gesagt werden, dass die Ziele der Arbeit größtenteils umgesetzt wurden und der Prototyp einer Anwendung zum Zeichnen von Entity-Relationship-Diagrammen realisiert wurde. Diese Anwendung wird in Zukunft als Download innerhalb der Lernplattform edb angeboten. Sie steht somit nicht nur für die Veranstaltung Datenbanken an der Fachhochschule Köln - Campus Gummersbach zur Verfügung, sondern eventuell auch einer größeren Community von Datenbankentwicklern oder anderen Universitäten, an denen das Thema Datenbanken unterrichtet wird.

12 Literaturverzeichnis

[ant] *Anti Patterns Catalog*. http://c2.com/cgi/wiki?AntiPatternsCatalog, Abruf: 10.02.2014

[Apaa] Apache Software Foundation (Hrsg.): *Apache Log4j 1.2*. Apache Software Foundation, http://logging.apache.org/log4j/1.2/, Abruf: 10.02.2014

[Apab] The Apache Software Foundation (Hrsg.): *Apache Maven Project*. The Apache Software Foundation, http://maven.apache.org, Abruf: 10.02.2014

[CA] CA Inc. (Hrsg.): *CA ERwin Data Modeler*. CA Inc., http://erwin.com/products/data-modeler, Abruf: 17.01.2014

[Che76] Chen, Peter: The Entity-Relationship Model - Toward a Unified View of Data / Massachusetts Institute of Technology. Version: 1976. http://csc.lsu.edu/news/erd.pdf, Abruf: 22.01.2014. 1976. – Forschungsbericht

[Dun 9] Dunkel, Jürgen: *Softwarearchitektur für die Praxis*. Springer, 2003, ISBN: 3-540-00221-9

[Ecl] The Eclipse Foundation (Hrsg.): *Eclipse Homepage*. The Eclipse Foundation, http://www.eclipse.org, Abruf: 10.02.2014

[fab] fabFORCE (Hrsg.): *DBDesigner Homepage*. fabFORCE, http://www.fabforce.net/dbdesigner4/, Abruf: 17.01.2014

[FF 6] Freeman, Eric ; Freeman, Elisabeth: *Head First Design Patterns*. O'Reilly Media, Inc., 2004, ISBN: 978-0-596-00712-6

[FH-] FH-Würzburg (Hrsg.): *Endlicher Automat*. FH-Würzburg, http://www.iwiki.de/wiki/index.php/Endlicher_Automat, Abruf: 25.01.2014

[Fow] Fowler, Martin: *Inversion of Control Containers and the Dependency Injection pattern*, http://martinfowler.com/articles/injection.html, Abruf: 10.02.2014

[FWBRB66] Faeskorn-Woyke, Heide ; Bertelsmeier, Birgit ; Riemer, Petra ; Bauer, Elena: *Datenbanksysteme: Theorie und Praxis mit SQL2003, Oracle und MySQL*. Addison-Wesley, 2007, ISBN: 3827372666

[GHJV 8] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2011, ISBN: 978-3-8273-3046-8

[Gla] Glassfisch Community (Hrsg.): *JAXB Homepage*. Glassfisch Community, https://jaxb.java.net, Abruf: 19.01.2014

[GLK] Gawenda, Damian ; Liss, Nico ; Kasper, Andre ; Fachhochschule Köln - Campus Gummersbach (Hrsg.): *eLearning Portal der FH-Köln, Campus Gummersbach*. Fachhochschule Köln - Campus Gummersbach, http://edb.gm.fh-koeln.de, Abruf: 17.01.2014

[Gre] Greer, Derek: *The Art of Seperation of Concerns*, http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/, Abruf: 25.01.2014

[IBM] IBM Corporation (Hrsg.): *IBM DB2 database software*. IBM Corporation, http://www-01.ibm.com/software/data/db2/, Abruf: 25.01.2014

[Int] International Organization for Standardization (Hrsg.): *ISO/IEC 25010:2011 - System and software engineering –Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. International Organization for Standardization, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733

[ITW] ITWissen - Lexikon (Hrsg.): *Framework Definition*. IT-Wissen - Lexikon, http://www.itwissen.info/definition/lexikon/Framework-framework.html, Abruf: 19.01.2014

[JGra] JGraph Ltd (Hrsg.): *JGraph Forum*. JGraph Ltd, http://forum.jgraph.com, Abruf: 19.01.2014

[JGrb] JGraph Ltd (Hrsg.): *JGraphX GitHub Page*. JGraph Ltd, https://github.com/jgraph/jgraphx, Abruf: 19.01.2014

[JGrc] JGraph Ltd (Hrsg.): *JGraphX (JGraph6) User Manual*. JGraph Ltd, http://jgraph.github.io/mxgraph/docs/manual_javavis.html, Abruf: 19.01.2014

[jun] *JUnit Homepage*. : *JUnit Homepage*, http://junit.org, Abruf: 10.02.2014

[KP09] Kasper, Andre ; Phillip, Jan: *Visualisierung der Abhängigkeiten von Datenbankobjekten*, Fachhochschule Köln, Diplomarbeit, 2009. http://www.visualdependencies.de/builds/DA-2009-philipp-kasper-v1.1.pdf, Abruf: 19.01.2014

[Mar] Martin, Robert C. ; Object Mentor Inc (Hrsg.): *The Single Responsibility Principle*. Object Mentor Inc, http://www.objectmentor.com/resources/articles/srp.pdf, Abruf: 25.01.2014

[Mar59] Martin, James: *Information Engineering Book II: Planing and Analysis*. Prentice Hall, 1989, ISBN: 978-0134648859

[Mic] Microsoft Corporation (Hrsg.): *Microsoft Visio Homepage*. Microsoft Corporation, http://office.microsoft.com/de-de/visio/, Abruf: 17.01.2014

[Obj] Object Management Group (Hrsg.): *Unified Modeling Language*. Object Management Group, http://www.omg.org/spec/UML/, Abruf: 17.01.2014

[OFNa] O'Madadhain, Joshua ; Fisher, Danyel ; Nelson, Tom: *Java Universal Network/Graph 2.0 Framework Tutorial*, http://www.grotto-networking.com/JUNG/JUNG2-Tutorial.pdf, Abruf: 19.01.2014

[OFNb] O'Madadhain, Joshua ; Fisher, Danyel ; Nelson, Tom: *Java Universal Network/Graph Framework*, http://jung.sourceforge.net, Abruf: 08.12.2013

[Ope] Open Source Initiative (Hrsg.): *Open Source Initiative Homepage*. Open Source Initiative, http://opensource.org, Abruf: 19.01.2014

[Oraa] Oracle Corporation (Hrsg.): *Annotations*. Oracle Corporation, http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html, Abruf: 25.01.2014

[Orab] Oracle Corporation (Hrsg.): *Data Definition Statements*. Oracle Corporation, http://dev.mysql.com/doc/refman/5.1/en/sql-syntax-data-definition.html, Abruf: 19.01.2014

[Orac] Oracle Corporation (Hrsg.): *Enum Types*. Oracle Corporation, http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html, Abruf: 25.01.2014

[Orad] Oracle Corporation (Hrsg.): *How to Use File Choosers*. Oracle Corporation, http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html, Abruf: 25.01.2014

[Orae] Oracle Corporation (Hrsg.): *Javadoc Tool*. Oracle Corporation, http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html, Abruf: 19.01.2014

[Oraf] Oracle Database Administration Resources (Hrsg.): *Data Definition Language (DDL) Statements*. Oracle Database Administration Resources, http://www.oracle-dba-online.com/sql/oracle_data_definition_language.htm, Abruf: 19.01.2014

[Piv] Pivotal Software Inc (Hrsg.): *Spring Framework Homepage*. Pivotal Software Inc, http://spring.io, Abruf: 10.02.2014

[Ram] Rampl, Hansjörg: *Usability Engineering*, http://www.handbuch-usability.de/usability-engineering.html, Abruf: 10.02.2014

[Scr] Scrum Alliance Inc. (Hrsg.): *Scrum Homepage*. Scrum Alliance Inc., http://www.scrumalliance.org, Abruf: 17.01.2014

[Sun] Sun Microsystems (Hrsg.): *MySQL Workbench Homepage*. Sun Microsystems, http://www.mysql.de/products/workbench/, Abruf: 10.02.2014

[wika] *Wikipedia*. http://www.wikipedia.org, Abruf: 17.01.2014

[Wikb] Wikipedia (Hrsg.): *Entity-Relationship-Modell*. Wikipedia, http://de.wikipedia.org/wiki/Entity-Relationship-Modell, Abruf: 17.01.2014

[Wol20] Wolff, Eberhard: *Spring 3: Framework für die Java-Entwicklung*. dpunkt, 2010, ISBN: 978-3898645720

[Wor] World Wide Web Consortium (Hrsg.): *Extensible Markup Language(XML)*. World Wide Web Consortium, http://www.w3.org/XML/, Abruf: 19.01.2014

[yWo] yWorks GmbH (Hrsg.): *yFiles for Java*. yWorks GmbH, http://www.yworks.com/de/products_yfiles_about.html, Abruf: 27.01.2014

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 17. Februar 2014